Gaussian Processes for Text Regression

Daniel Emilio Beck

Faculty of Engineering
Department of Computer Science
University of Sheffield

A thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

June 2017

Abstract

Text Regression is the task of modelling and predicting numerical indicators or response variables from textual data. It arises in a range of different problems, from sentiment and emotion analysis to text-based forecasting. Most models in the literature apply simple text representations such as bag-of-words and predict response variables in the form of point estimates. These simplifying assumptions ignore important information coming from the data such as the underlying uncertainty present in the outputs and the linguistic structure in the textual inputs. The former is particularly important when the response variables come from human annotations while the latter can capture linguistic phenomena that go beyond simple lexical properties of a text.

In this thesis our aim is to advance the state-of-the-art in Text Regression by improving these two aspects, better uncertainty modelling in the response variables and improved text representations. Our main workhorse to achieve these goals is Gaussian Processes (GPs), a Bayesian kernelised probabilistic framework. GP-based regression models the response variables as well-calibrated probability distributions, providing additional information in predictions which in turn can improve subsequent decision making. They also model the data using kernels, enabling richer representations based on similarity measures between texts.

To be able to reach our main goals we propose new kernels for text which aim at capturing richer linguistic information. These kernels are then parameterised and learned from the data using efficient model selection procedures that are enabled by the GP framework. Finally we also capitalise on recent advances in the GP literature to better capture uncertainty in the response variables, such as multi-task learning and models that can incorporate non-Gaussian variables through the use of warping functions.

Our proposed architectures are benchmarked in two Text Regression applications: Emotion Analysis and Machine Translation Quality Estimation. Overall we are able to obtain better results compared to baselines while also providing uncertainty estimates for predictions in the form of posterior distributions. Furthermore we show how these models can be probed to obtain insights about the relation between the data and the response variables and also how to apply predictive distributions in subsequent decision making procedures.

Nothing in life is to be feared, it is only to be understood.

Marie Curie

Acknowledgements

Doing a PhD is like riding a roller coaster. Sometimes scary, sometimes fun, but it is utmost a thrilling experience. All the ups and downs that happen during the ride give you a mix of emotions, ranging from moments of exhilarating joy to strong feelings of regret. And in the end, it is mostly up to you to make sure you can cope with those and reach the end in one piece. It is similar for a PhD: to get the title you have to prove you can be an independent researcher, make your own choices and deal with the outcomes, whatever they are.

However, in a roller coaster you do not ride alone, you share the cart with others. There are also technicians and managers to make sure the ride runs smoothly. And finally, some people stay outside watching and cheering up for you. These words are dedicated to these people.

Sharing the front row with me and holding my hand when I needed were my supervisors, Lucia Specia and Trevor Cohn. Lucia gave invaluable guidance in the beginning by putting me right into research mode from day one. She also opened many doors and paved the way for me to showcase my work to many people in the field. And she was always ready and prepared to put out the uncountable fires that I started during my journey. Trevor was like a personal Machine Learning encyclopedia. His mix of technical and creative skills was not only fundamental for my PhD but pretty much shaped me as a researcher. The best gift I got from them, however, was their sheer trust on my abilities to pursue my own research avenues. Many times I thought they were foolish for giving me so much freedom and that I was not ready to be on my own. But this thesis is proof they were right from the very beginning. Thank you.

At the control room, waiting for me to arrive and checking if I endured the ride, were my examiners, Andreas Vlachos and Xavier Carreras. Andreas was a great source of knowledge and friendship in Sheffield and I am happy I was able to share some of my PhD time with him. Xavier taught me fundamental concepts on structured prediction at the Lisbon ML Summer School and I was glad when he accepted to be part of my panel. Much of this thesis' contents were greatly improved by their comments and having their seal of approval is something I am proud of.

In the cart with me and sharing many of the thrills were my colleagues at the NLP group in Sheffield. In particular, I would like to thank Carol Scarton, Fred Blain, Gustavo Paetzold, Kashif Shah, Wilker Aziz, Johann Petrak, David Steele, Karin Sim Smith, Michal Lukasik, Makis Lampouras, Ahmet Aker, Chiraag Lala, Roland Roller and Pranava Madhyastha. Thank you for all the research and non-research conversations, either in the lab or in the pub. I feel very lucky to have been part of this amazing and heartwarming crew.

The journey also involved spending some time in other rides as well. In Sheffield, I benefited a lot from the Gaussian Process Summer School and talks from the ML group at SITRAN. Special thanks to James Hensman, who pointed me to Warped GPs, which became a fundamental part of this thesis. I also had the opportunity to attend the Lisbon ML Summer School twice, first as a student and later as a monitor. The learning and networking opportunities I got from the school helped me greatly during the PhD. Between 2014 and 2015 I spent six months at the University of Melbourne, in Australia. Thanks again Trevor for inviting me and Tim Baldwin and all members of the group for making me feel at home. Finally, the internship at SDL Cambridge showed me how is it like to do research in a corporate environment. Many thanks to Adrià de Gispert, Gonzalo Iglesias, Rory Waite and Bill Byrne for the experience.

Cheering me up from outside were my friends and family, who were always there for me even though physically scattered around the world. My years in Sheffield would not be the same without Xosé, Angela, Mariam, Andrey, Emily, Eva, Jess and many others. Furthermore, thanks to all my friends in Brazil and elsewhere who supported me from far away. And I would not ever be able to finish this journey without the love of my family, especially my brothers Dary and Diego and my mother, Maria Teresa. Thank you for letting me wander, I love you.

Last, but not least, thanks to you, the reader. I hope this thesis provide you the knowledge you seek. And if you are a PhD student, enjoy the ride!

Oh yeah, roller coasters need money to be built and maintained. This work was funded by CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brazil (Science Without Borders No. 237999/2012-9).

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Making sense of natural language has been one of the most elusive goals of Artificial Intelligence. Human texts are a rich source of information which can be useful in numerous fields. For instance, sentences in natural language can convey sentiment information due to the existence of semantic links between words and emotional states. Accurate models that are able to express these links can bring insights about how these links are created in the first place. These can guide linguistic and psycholinguistic studies that aim at understanding the mechanisms from where humans generate language.

On a more applied perspective, the advent of Natural Language Processing (NLP) systems made machine-generated text widespread. Perhaps the most well known application is Machine Translation (MT), where users rely on automatically generated translations for many different purposes, from simple gisting to aiding human translation projects in complex workflows. A common use of MT is in *post-editing* environments, where professional translators are given machine translated texts as a starting point for the actual translation task. While this procedure tends to enhance productivity in general (Plitt and Masselot, 2010) it is a known fact that MT systems are not perfect and still prone to many errors. This not only affects the post-editing performance itself but can also hurt the overall experience of the human translators involved in the process. In this case, having a precise quality measurement of the MT-generated text can help to filter badly translated segments or to estimate project costs, among other uses.

The examples showed above require predictions in the form of indicators that summarise some specific aspect from a text. Many of these indicators come in the form of continuous *response variables*. Sentiment information can be provided in a numerical scale showing how strongly positive (or negative) a text is. Machine translated sentences can be measured in terms of how long a human translator will take to post-edit it to an acceptable quality.

Predicting such response variables is the goal of Text Regression, which is the main focus of this thesis.

Regression models are widespread in Machine Learning (ML), more specifically in supervised learning settings. Here the main goal is to *predict* a response variable from unseen data. Depending on the task, the response variables can exhibit varying degrees of *uncertainty*. Some regression models, especially ones derived from Bayesian theory, can capture this uncertainty in the form of well-calibrated predictive distributions, which in turn provide extra indicators for analysis and decision making.

While uncertainty-aware regression models are well studied in the literature, this aspect is mostly ignored in previous work in Text Regression. The majority of the current models only provide single point estimates as predictions, which is problematic because even if the model can generalise well there is no reason to believe the training labels are fully reliable. This is particularly important when they come from manual annotations, either explicitly or implicitly. Humans exhibit a large diversity of biases that directly affect these annotations, especially considering the fine-grained nature of response variables. In these settings, models should ideally be aware of these biases and propagate this information to the predictions.

Another aspect which is paramount in Text Regression is how to *represent* the textual input. In most regression models the data is represented as a set of manually designed *features* that reflect some kind of prior knowledge about the problem and/or the data. However, for text data it is usually not clear what kind of features are good predictors. Alternative solutions need to be used for representing text in a ML-friendly format.

A common approach for this problem is to encode each word as vector in a real valued space and generate a feature representation for the whole segment by combining the corresponding word vectors in a sensible way. If these vectors are one-hot encodings for each word and the combination is a sum the result is a *bag-of-words* (BOW) representation. This can be enhanced by weighted schemes such as TF-IDF (Jurafsky and Martin, 2000, Sec. 17.3), which assigns higher weights for more discriminative words. An alternative to sparse encodings is to employ dense word vectors obtained via *distributional* representations (Deerwester et al., 1990; Mikolov et al., 2013; Pennington et al., 2014), which aim at capturing linguistic properties of words based on their context. In this case, the overall segment-level representation can be obtained by averaging word vectors instead of summing them.

The main drawback of these approaches is that they are agnostic to word order, which also contains important linguistic information. Some dependencies can be captured by considering higher order word sequences such as bigrams or by applying convolutions to vector sequences.

However, these approaches are restricted to contiguous sequences for computational purposes and fail to capture complex phenomena, like long distance interactions, for instance.

1.1 Thesis Aims

The aim of this thesis is to improve Text Regression with respect to two aspects:

Better uncertainty modelling in the response variables. This will result in models that provide more information for decision making and take into account possible biases coming from annotations.

Improved text representations. With this, we expect to capture more diverse linguistic phenomena when compared to simpler approaches such as BOW.

1.1.1 Method overview

Our main workhorse for reaching this thesis' aims is Gaussian Processes (GPs) (Rasmussen and Williams, 2006). GPs combine two major aspects which we explore to solve the problems cited at the beginning of this chapter:

- Being a Bayesian framework, GPs are very powerful in modelling uncertainty. This uncertainty is propagated in an end-to-end fashion and predictions are provided as well calibrated probability distributions instead of single point estimates.
- GPs are *kernelised*, which means they rely on *kernels* to model input data. This opens the door for better text representation because kernels lift the restriction of defining texts in terms of a set of features: they can be defined as similarity functions between the texts themselves.

1.1.2 Scope

While the methods presented in this thesis can in principle be employed in a broad range of regression problems, in order to achieve our aim we apply our models to a set of benchmark tasks. All scope restrictions we assume in this study directly reflect the nature of the data used in these tasks.

Our benchmarks comprise both human and machine generated text, where the size of each textual input is in the form of a natural language sentence. We do not apply our models

to long bodies of text such as paragraphs or whole documents. We also focus on benchmarks where available data is scarce, in the order of thousands of instances.

Finally, although they could be applied to any language where the employed resources are available, most of our proposed benchmarks use English data. One of our tasks has an inherent bilingual component and we do perform experiments in a range of language pairs but we do not deeply assess how different languages affect the behaviour of our models.

1.2 Thesis Structure

The first two chapters provide a theoretical foundation for the models that will be used later and reviews previous work using these models in the context of Text Regression and NLP in general. Chapter 2 introduces Gaussian Processes, including GP-based regression models and hyperparameter optimisation, a central aspect that allows efficient model selection.

Chapter 3 gives our first major contribution of this thesis, new kernels for dealing with structured data such as strings and trees. We give an overview of previous work which developed the theory behind such kernels and extend them to allow efficient hyperparameter optimisation within GP-based models. We further explore this idea to show how it enables more complex parameterisations that enrich the capacity of these kernels.

The next two chapters correspond to the applications where we assess our proposed methods. Emotion Analysis (EA) is the focus of Chapter 4, where the goal is to model latent emotions present in textual data. Specifically, we propose two approaches for this task. One is based on string kernels and uses dense word vectors to capture similarities between words. The second is a model that incorporate response variables from multiple emotions in a joint manner.

In Chapter 5 we address the task of Machine Translation Quality Estimation (QE), which aims at predicting the quality of automatically translated text segments. We focus on predicting post-editing time at the sentence level and propose two approaches for this problem. The first uses GP extensions that are able to learn non-Gaussian distributions, in order to better capture the uncertainty in the response variables. We further explore this idea by showing two applications for the predictive distributions generated by our models. The second approach proposes the use of tree kernels to better capture syntactic phenomena while also moving away from the usual feature-based models employed in this task.

Finally, Chapter 6 summarises the findings and gives directions for future work.

1.3 Contributions

In this section we state the main research contributions of this thesis.

Text Regression models that provide well-calibrated predictive distributions. These models capitalise on recent advances made in the GP literature which learn the support of these distributions and leverage correlations when multiple response variables are present.

A novel kernel framework for textual data. This framework combines previous work on kernels for discrete structures with the efficient model selection procedures provided by GPs. We also explore this idea to enhance these kernels with a set of novel parameterisations, which further enhance their modelling power. The proposed framework can be applied to either surface texts (strings) or to preprocessed data in the form of syntactic trees.

New methods for Emotion Analysis. We devise two new methods for this task. The first one is based on combining string kernels with GPs, where we show it can give better performance than linear methods trained on averaged embeddings. Our second approach uses multi-task GPs, which are able to learn existing correlations and anti-correlations between emotions. Experiments show that this approach gives state-of-the-art predictive performance while also enhancing interpretability by probing into the learned covariance matrices.

Novel approaches for Machine Translation Quality Estimation. We investigate two approaches for this task. The first one use models that are able to incorporate non-Gaussian response variables. We use them to predict post-editing time for machine translated sentences and show how to employ the predictive distributions in two applications, one based on asymmetric risks and one that uses active learning to accelerate model performance. Our second method takes into account complexity and fluency aspects of each sentence pair by encoding them as syntactic trees. We employ these trees as inputs for a model based on combining tree kernels with GPs, giving promising results.

1.4 Published Material

Some of the material presented in this thesis overlaps with publications in peer-reviewed venues, shown below:

- The material in Chapter 3 contains ideas from a thesis proposal paper at the ACL 2014 Student Research Workshop (Beck, 2014) and from a paper on tree kernels and GPs, published in a TACL issue in 2015 (Beck et al., 2015). The TACL paper also inspired the tree kernel experiments in Chapter 5.
- Some of the Emotion Analysis experiments in Chapter 4 overlap with an EMNLP 2014 manuscript (Beck et al., 2014a).
- Much of the content in Chapter 5 originated from two Quality Estimation shared task submissions at the WMT workshop in 2013 and 2014 (Beck et al., 2013a, 2014b). The uncertainty evaluation and asymmetric risk experiments appeared first in a CONLL 2016 paper (Beck et al., 2016) while the active learning experiments were inspired by an ACL 2013 paper (Beck et al., 2013b).

Chapter 2

Gaussian Processes

In this chapter we introduce the theoretical grounds for Gaussian Processes and uncertainty evaluation. We start by revisiting linear regression in Section 2.1, including Bayesian models which are able to infer probability distributions over parameters and predictions. Then we motivate the need for non-linear regression in Section 2.2 and show how the initial model can be extended to this setting by introducing kernels. In Section 2.3 we present Gaussian Processes by explaining how they combine ideas from Bayesian theory and kernel methods, while also introducing some of the extensions that will be used throughout this thesis. Finally, in Section 2.4 we discuss how to evaluate models that are able to produce uncertainty estimates and wrap up in Section 2.5.

The derivations in Sections 2.1 and 2.2 are inspired by Bishop (2006), Murphy (2012) and Shawe-Taylor and Cristianini (2004), while Section 2.3 is based mainly on Rasmussen and Williams (2006).

2.1 Linear Regression

Linear regression is arguably the main workhorse in ML, providing the basis for many other advanced methods in the field. We give a brief summary of its foundations, which will provide useful for better understanding of the models we will introduce later.

Consider a set of data points $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, with n being its size and d being the dimensionality of each vector $\mathbf{x}$. As the name implies, linear regression aims at finding a linear mapping $f(\mathbf{x})$ between each input and its response variable y:

$$f(\mathbf{x}) = w_0 + \sum_{i=1}^{d} x_i w_i + \epsilon, \tag{2.1}$$

where w_0 is the intercept term and ϵ is the error between the model predictions and true response variable. A common notational trick is to prepend a constant 1 to each input, redefining it as $[1, \mathbf{x}]$. This allows us to implicitly combine the intercept term into a more general form:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + \epsilon. \tag{2.2}$$

We assume this notational trick throughout the rest of this chapter.

The error term ϵ is usually assumed to be Gaussian: $\epsilon \sim \mathcal{N}(0, \sigma^2)$. By plugging it into Equation 2.2 and assuming fixed variance for all inputs, we can express the model as a conditional probability density:

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mathbf{x}^T\mathbf{w}, \sigma^2), \tag{2.3}$$

where $\boldsymbol{\theta} = \{\mathbf{w}, \sigma^2\}$ is an umbrella term that contains all the model parameters.

Given a dataset $\mathcal{D}$, the most common criterion to fit the model parameters is via *Maximum Likelihood Estimate* (MLE):

$$\hat{\boldsymbol{\theta}}_{MLE} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\mathcal{D}|\boldsymbol{\theta}), \tag{2.4}$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}), \tag{2.5}$$

where Equation 2.5 follows from assuming the data is independent and identically distributed (iid). By inserting the definition of a Gaussian on Equation 2.5 we reach the following solution:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left[(2\pi\sigma^2)^{-\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - \mathbf{x}_i^T\mathbf{w})^2 \right) \right], \tag{2.6}$$

$$= \sum_{i=1}^{n} \log(2\pi\sigma^2)^{-\frac{1}{2}} - \frac{1}{2\sigma^2}(y_i - \mathbf{x}_i^T\mathbf{w})^2, \tag{2.7}$$

$$= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\mathrm{SSE}(\mathbf{w}) \tag{2.8}$$

where SSE means *sum of squared errors*.

Following Equation 2.8, we can see that the MLE for $\mathbf{w}$ corresponds to the one that minimizes $\mathrm{SSE}(\mathbf{w})$. To do this, first we rewrite SSE in matrix form:

$$\mathrm{SSE}(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}). \tag{2.9}$$

Then, we take the gradient vector with respect to $\mathbf{w}$ and set it to zero:

$$\frac{\partial \text{SSE}(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})^T}{\partial \mathbf{w}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + (\mathbf{y} - \mathbf{X}\mathbf{w})^T \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}} \tag{2.10}$$

$$0 = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + (\mathbf{y} - \mathbf{X}\mathbf{w})^T(-\mathbf{X}) \tag{2.11}$$

$$\hat{\mathbf{w}}_{ML} \triangleq (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.12}$$

The equation above is known as *normal equation* and corresponds to the MLE solution for $\mathbf{w}$. At prediction time, we can obtain the response variable for a new, unseen instance $\mathbf{x}_*$ by just taking the dot product: $y_* = \mathbf{x}_*^T \hat{\mathbf{w}}_{ML}$.

Figure 2.1 shows an example of a linear regression fit to a 15 point dataset. The data was generated by sampling points to a line (represented as "truth" in the plot) and adding random Gaussian noise. We can see that in this case the model is very close to the truth since the linear assumption holds for our generated data, even in the presence of noise.

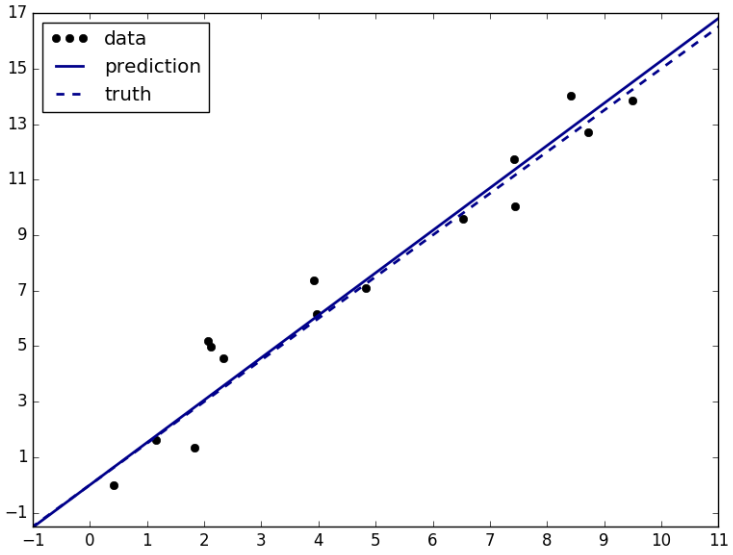


Fig. 2.1 Linear regression example. The prediction line was obtained by solving the normal equation with respect to the generated data points.

2.1.1 Dealing with overfitting through priors

In real world data is usually not as well behaved as shown in Figure 2.1. For instance, sometimes we can have outliers that do not follow the trend observed for most data points. In

this situation, linear regression can *overfit*, in a sense that it artificially increases its capacity[1] in order to accommodate these outliers.

Overfitting in linear regression happens when the weight parameters end up being abnormally large. A simple way to tackle this problem is by placing a Gaussian prior $\mathcal{N}(0, \tau^2)$ on the weights, which will encourage them to be small. Then we can use a *Maximum a Posteriori* (MAP) estimate of the parameters:

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmax}}[\log p(\mathcal{D}|\mathbf{w}, \sigma^2) + \log p(\mathbf{w})], \tag{2.13}$$

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \left[\sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) + \sum_{j=1}^{d} \log p(w_j|0, \tau^2) \right]. \tag{2.14}$$

It is possible to show that finding the optimal MAP weights result in the same objective as MLE with an additional term corresponding to the squared norm of $\mathbf{w}$:

$$J(\mathbf{w}) = \frac{1}{N}\text{SSE}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2, \tag{2.15}$$

where $\lambda = \sigma^2/\tau^2$ is usually referred as the *regularisation* hyperparameter[2]. By setting the gradient vector to 0 and solving for $\mathbf{w}$ we arrive at the following solution:

$$\hat{\mathbf{w}}_{MAP} \triangleq (\lambda \mathbf{I}_d + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.16}$$

In the literature this technique is known as *ridge regression*. The norm of $\mathbf{w}$ can be interpreted as a measure of model capacity as it encourages small values for $\mathbf{w}$. Figure 2.2 shows an example where we have two outliers. We can see that ridge regression is able to reach a better fit when compared to standard linear regression.

If we want a measure of uncertainty over predicted response variables we can obtain a MAP estimate for σ^2:

$$\sigma^2_{MAP} \triangleq \frac{1}{N}\sum_{i=1}^{n}(\mathbf{x}_i^T\hat{\mathbf{w}}_{MAP} - y_i)^2 \tag{2.17}$$

[1] Here we define capacity as the *effective model complexity*, as coined by (Bishop, 2006, Sec. 3.1.4). In this sense, the capacity corresponds not only to the number of parameters in a model but also to the effective range these parameters can take. See (Bishop, 2006, Sec. 3.2) for more details on this definition. While we could use "complexity" instead of "capacity", we use the latter so it is not confused with *algorithmic* complexity when describing the models and their corresponding training methods.

[2] In the Bayesian literature, an hyperparameter is a parameter of a prior distribution. Here we loosely extend this concept to reflect any quantities that lie in a higher level in the model hierarchy.

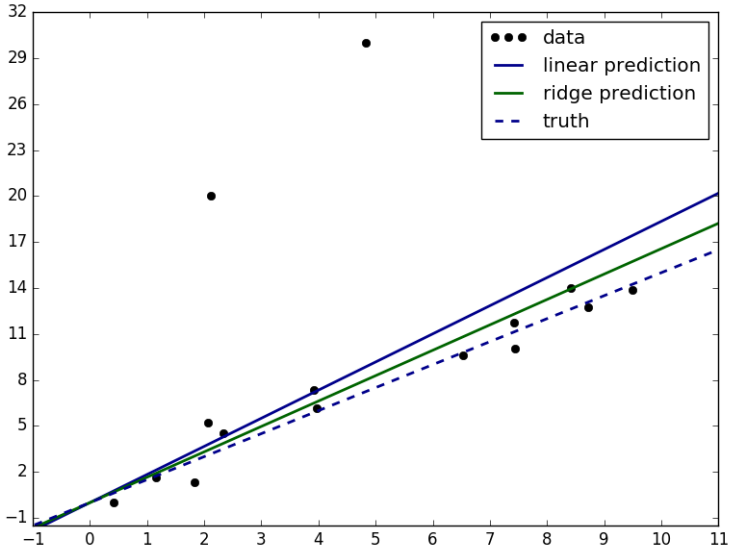


Fig. 2.2 Ridge regression example. Here we artificially introduced two outliers to the same dataset shown on Figure 2.1. We can see that ridge regression is able to mitigate the outliers influence due to regularisation.

The main issue with this estimate is that it is constant regardless of the value of $\mathbf{x}_i$. Ideally, we want uncertainty estimates to be lower if an unseen instance is close to the training points and higher otherwise.

To reach this behaviour we can keep the same prior over the weights but instead of finding a point estimate solution we should follow a Bayesian approach and integrate over all possible values of $\mathbf{w}$:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|\mathbf{x}_*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{y}) d\mathbf{w}, \tag{2.18}$$

where we omit σ^2 and τ^2 from the conditionals for clarity. This model is known as Bayesian linear regression. For simplicity we assume σ^2 and τ^2 are known in the next derivations.

The first step in Equation 2.18 is to obtain the posterior over $\mathbf{w}$. For simplicity, we follow the same setting as in ridge regression and assume $\mathcal{N}(0, \tau^2)$ as the prior for each individual weight[3]. Since both the prior and likelihood are Gaussians, the posterior over $\mathbf{w}$ ends up

[3]Derivations for arbitrary multivariate Gaussian priors for $\mathbf{w}$ can be found in (Bishop, 2006, Sec. 3.3) and (Murphy, 2012, Sec 7.6).

being another Gaussian due to conjugacy:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}, \sigma^2) \propto \mathcal{N}(\mathbf{w}|\mathbf{0}, \tau^2\mathbf{I}_d)\mathcal{N}(\mathbf{y}|\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}_n) = \mathcal{N}(\mathbf{w}|\mathbf{w}_n, \mathbf{V}_n), \tag{2.19}$$

$$\mathbf{w}_n = \frac{1}{\sigma^2}\mathbf{V}_n\mathbf{X}^T\mathbf{y}, \tag{2.20}$$

$$\mathbf{V}_n = \sigma^2\left(\frac{\sigma^2}{\tau^2}\mathbf{I}_n + \mathbf{X}^T\mathbf{X}\right)^{-1}. \tag{2.21}$$

Notice that the posterior mean is the same as $\hat{\mathbf{w}}_{MAP}$ for ridge regression. Plugging the posterior into Equation 2.18 lets us obtain the final distribution for the predictive posterior, which is also a Gaussian:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}, \sigma^2) = \int \mathcal{N}(y_*|\mathbf{x}_*^T\mathbf{w}, \sigma^2)\mathcal{N}(\mathbf{w}|\mathbf{w}_n, \mathbf{V}_n)d\mathbf{w}, \tag{2.22}$$

$$= \mathcal{N}(y_*|\mathbf{x}_*^T\mathbf{w}_n, \sigma_n^2(\mathbf{x}_*)), \tag{2.23}$$

$$\sigma_n^2(\mathbf{x}_*) = \sigma^2 + \mathbf{x}_*^T\mathbf{V}_n\mathbf{x}_*. \tag{2.24}$$

While in ridge regression we assume the predictive variance is simply σ^2, in this model it also depends on $\mathbf{x}_*^T\mathbf{V}_n\mathbf{x}_*$, which include the variance of the parameters. The resulting behaviour is that points far from the original training data will have more spread distributions, which translate into higher uncertainty over the response variable.

Figure 2.3 shows how ridge and Bayesian linear regression differ in their uncertainty estimates. While the posterior mean is the same for both models, the Bayesian model is able to increase the uncertainty for predictions that are far from the training data.

2.1.2 Linear models in NLP

Linear models are vastly used in multiple NLP applications. In regression, Joshi et al. (2010) introduced a model for movie revenue forecasting based on critic reviews. They combine a bag-of-ngrams representation with an Elastic Net regulariser (Zou and Hastie, 2005), which aims at finding sparse and small weights (i.e. $w = 0$ for most dimensions). This allows the algorithm to effectively select only a subset of ngrams which are most discriminative. This, in turn, can speed-up predictions and facilitate feature analysis. Variants of linear regression were also employed in previous work in Quality Estimation (Quirk, 2004) and Emotion Analysis (Mihalcea and Strapparava, 2012).

It is also possible to employ linear models in text classification, which is arguably much more common than regression in previous work (Lewis et al., 1996; Sebastiani, 2002; Wang

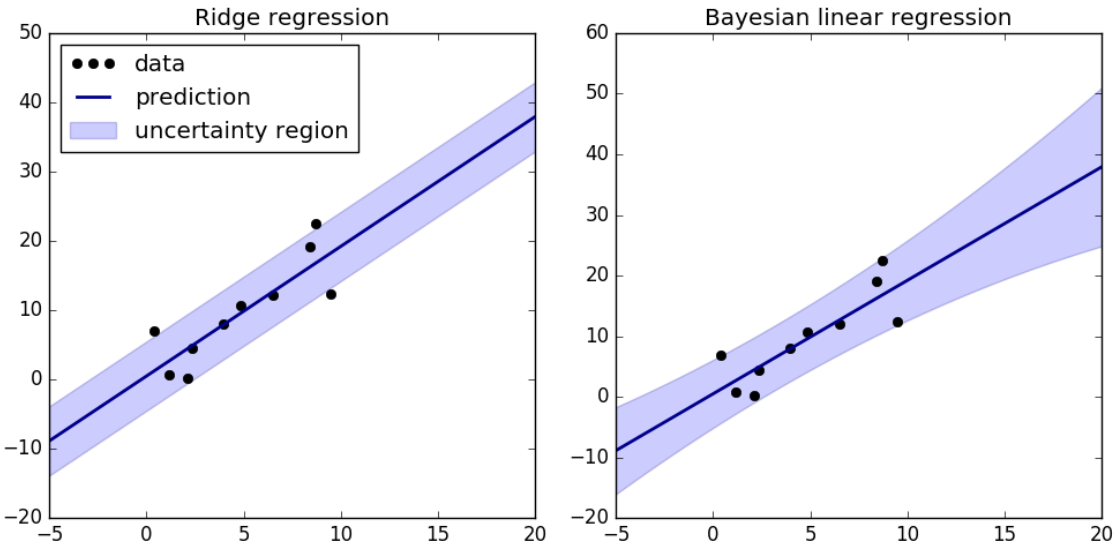


Fig. 2.3 Ridge and Bayesian linear regression with their respective uncertainty intervals, corresponding to $\pm 2 \times \sigma(\mathbf{x})$.

and Manning, 2012, *inter alia)*. Logistic regression is the most natural extension of the linear regression model to (binary) classification since it basically changes the likelihood from a Gaussian to Bernoulli. However, historically other methods were more commonly applied in text classification such as Naive Bayes or Perceptron. Finally, many structured prediction models used in NLP can be seen as extensions of linear models, such as Conditional Random Fields (Lafferty et al., 2001) and Structured Perceptron (Collins, 2002), with applications ranging from Part-of-Speech tagging (Toutanova et al., 2003) to syntactic parsing (Finkel et al., 2008), among others.

Although there is still interest in linear models due to their easy scalability (Agarwal et al., 2014; Joulin et al., 2016), assuming linear relationships between the inputs and the response variables can be limiting for many problems and datasets. In the next Section we will see how to extend these models to allow for non-linear modelling.

2.2 Beyond Linearity: Basis Function Expansions and Kernels

Linear regression is a very simple algorithm for training a model but it implies the existence of a linear mapping between the inputs and the outputs. If this assumption does not hold

then it can perform very poorly. However, we can still use linear regression to find a suitable function for this data if we extend the feature space accordingly.

In Figure 2.4 we show a dataset that has a quadratic behaviour. We can see that the linear fit is not ideal. But if we represent each input as a three-dimensional feature vector $\langle 1, x, x^2 \rangle$ we will be able to find a quadratic mapping between the input and the output using the same algorithm as before. In Figure 2.4 this results in the orange quadratic curve, which is a much better fit. This method is called a *basis function expansion* and it is useful when we want to find more complex patterns in data.

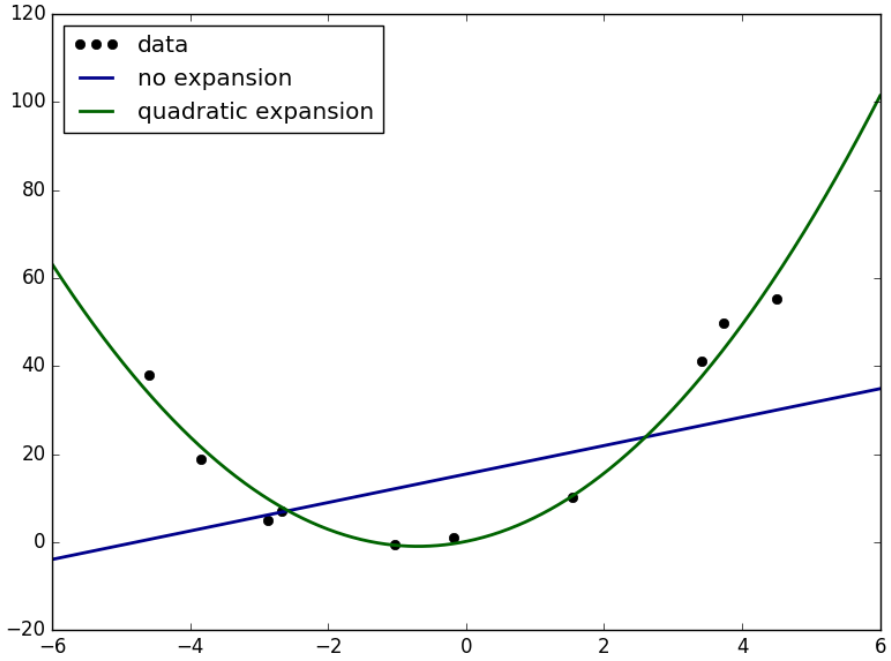


Fig. 2.4 A dataset with non-linear (quadratic) behaviour. While standard linear regression shows a poor fit, we can use a basis function expansion to find the quadratic curve.

The main downside of basis function expansions is that the feature space can rapidly explode: expanding a two-dimensional input into a cubic space results in a 15-dimensional feature vector. This means we reach a setting where $d \gg n$ and Equation 2.12 ends up being expensive to solve due to the matrix inversion. To deal with this we can actually rewrite it in

a different way. The main trick is to express $\mathbf{w}$ as a linear combination of the training points:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.25}$$

$$\mathbf{X}^T\boldsymbol{\alpha} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.26}$$

$$\boldsymbol{\alpha} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y} \tag{2.27}$$

$$= \mathbf{K}^{-1}\mathbf{y}. \tag{2.28}$$

where $\boldsymbol{\alpha}$ is a new set of weights, one per training input, and the matrix $\mathbf{K}$ is called *Gram matrix*, composed by the dot products of all training input combinations:

$$\mathbf{K} = \begin{pmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \langle \mathbf{x}_1, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \langle \mathbf{x}_2, \mathbf{x}_1 \rangle & \langle \mathbf{x}_2, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_2, \mathbf{x}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \langle \mathbf{x}_n, \mathbf{x}_2 \rangle & \cdots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{pmatrix}.$$

By definition, the Gram matrix is positive semi-definite (PSD)[4], since it arises as a square matrix of inner products of a set of vectors. It is a central part of the kernel methods we are going to show in this Section. To find $\boldsymbol{\alpha}$ we need to invert the Gram matrix, which has size $n \times n$. Prediction is done by plugging $\mathbf{w} = \mathbf{X}^T\boldsymbol{\alpha}$ into Equation 2.1:

$$y_* = \langle \mathbf{w}, \mathbf{x}_* \rangle = \left\langle \sum_{i=1}^{n} \alpha_i \mathbf{x}_i, \mathbf{x}_* \right\rangle = \sum_{i=1}^{n} \alpha_i \langle \mathbf{x}_i, \mathbf{x}_* \rangle = \mathbf{y}^T\mathbf{K}^{-1}\mathbf{k}_*, \tag{2.29}$$

where $\mathbf{k}_* = [\langle \mathbf{x}_1, \mathbf{x}_* \rangle, \langle \mathbf{x}_2, \mathbf{x}_* \rangle, \ldots, \langle \mathbf{x}_n, \mathbf{x}_* \rangle]$ and the last step results from using Equation 2.28. Notice that we do not use $\boldsymbol{\alpha}$ explicitly here, the prediction is done only considering the training response variables, the Gram matrix and $\mathbf{k}_*$.

We presented two different ways to perform linear regression: one in terms of $\mathbf{w}$ and another one in terms of $\boldsymbol{\alpha}$. The first form is called *primal* and the second one, *dual*. The key picture here is that if we have a dataset where $d \gg n$ performing linear regression in the dual form will be less expensive than in the primal form.

Being able to use of an arbitrary number of features in the training set lets us employ arbitrary basis function expansions. If we define an expansion as a mapping $\boldsymbol{\phi} : \mathbf{x} \in \mathbb{R}^d \rightarrow$

[4] A $d \times d$ matrix $\mathbf{M}$ is PSD if it is symmetric and $\mathbf{x}\mathbf{M}\mathbf{x}^T \geq 0$ for all $\mathbf{x} \in \mathbb{R}^d$.

$\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^D$, we can rewrite the Gram matrix as:

$$\mathbf{K} = \begin{pmatrix} \langle \boldsymbol{\phi}(\mathbf{x}_1), \boldsymbol{\phi}(\mathbf{x}_1) \rangle & \langle \boldsymbol{\phi}(\mathbf{x}_1), \boldsymbol{\phi}(\mathbf{x}_2) \rangle & \cdots & \langle \boldsymbol{\phi}(\mathbf{x}_1), \boldsymbol{\phi}(\mathbf{x}_n) \rangle \\ \langle \boldsymbol{\phi}(\mathbf{x}_2), \boldsymbol{\phi}(\mathbf{x}_1) \rangle & \langle \boldsymbol{\phi}(\mathbf{x}_2), \boldsymbol{\phi}(\mathbf{x}_2) \rangle & \cdots & \langle \boldsymbol{\phi}(\mathbf{x}_2), \boldsymbol{\phi}(\mathbf{x}_n) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}_1) \rangle & \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}_2) \rangle & \cdots & \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}_n) \rangle \end{pmatrix}.$$

Predictions can also be rewritten in a similar way:

$$\begin{aligned} f(\mathbf{x}_*) &= \mathbf{y}^T \mathbf{K}^{-1} \mathbf{k}_* \\ &= \mathbf{y}^T \mathbf{K}^{-1} [\langle \boldsymbol{\phi}(\mathbf{x}_1), \boldsymbol{\phi}(\mathbf{x}_*) \rangle, \langle \boldsymbol{\phi}(\mathbf{x}_2), \boldsymbol{\phi}(\mathbf{x}_*) \rangle, \ldots, \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}_*) \rangle]. \end{aligned} \tag{2.30}$$

The key observation here is that both training and prediction can be defined in terms of dot products between basis function expansions of the inputs. Depending on how we define such an expansion, we can define the dot product between two expanded inputs as a function of the *original* inputs, effectively *bypassing* the expansion. These functions are called *kernels* and this bypassing is referred in the literature as the *kernel trick*. Kernels are the core of the methods we show in this thesis.

A kernel is a function k that for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ satisfies:

$$k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle, \tag{2.31}$$

for some expansion $\boldsymbol{\phi}$. For example, consider $\mathcal{X}$ a two-dimensional input space and the following mapping:

$$\boldsymbol{\phi} : \mathbf{x} = (x_1, x_2) \rightarrow \boldsymbol{\phi}(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2). \tag{2.32}$$

If we calculate the kernel using this mapping, we get:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle && (2.33) \\ &= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2)(x'^2_1, x'^2_2 \sqrt{2} x'_1 x'_2) \rangle && (2.34) \\ &= x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2 x_1 x_2 x'_1 x'_2 && (2.35) \\ &= (x_1 x'_1 + x_2 x'_2)^2 && (2.36) \\ &= \langle \mathbf{x}, \mathbf{x}' \rangle^2. && (2.37) \end{aligned}$$

In this case, we can see the kernel trick in action: we can calculate the dot product between two expanded inputs without relying on explicit feature expansion. In fact, the kernel trick allows us to employ linear regression in arbitrarily large, even infinite dimensional feature spaces because we never have to actually expand the inputs into these spaces. This powerful property makes kernel-based methods desirable, mainly for two reasons:

- It requires fewer assumptions about the underlying function that we are trying to model. For example, instead of assuming a linear or a cubic function we can just assume that the function is smooth and use an appropriate kernel.
- It allow us to define kernels on structured objects (like strings or trees) instead of vectorial inputs, reducing the need for feature engineering. In these cases, kernels can be seen intuitively as a similarity metric between two objects. This metric could use a rich underlying feature space composed of smaller substructures, for example.

2.2.1 Kernel examples

Kernels are a well studied subject in the ML literature. We already saw an example which is the linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle = \mathbf{x}^T \mathbf{x}'$. Another example is the polynomial kernel:

$$k(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^g, \tag{2.38}$$

where c and g are hyperparameters. Equation 2.37 is an instance of a polynomial kernel with $c = 0$ and $g = 2$.

The linear and the polynomial kernels are sometimes referred as *dot-product* kernels since they are directly extensions of the dot product between inputs.[5] Another widely used class is the *stationary* kernels, which are invariant to translations in the input space since they rely only on differences between inputs, regardless of their magnitude. Probably the most well known stationary kernel is the *squared exponential* (SE):

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \mathbf{\Sigma} (\mathbf{x} - \mathbf{x}')\right). \tag{2.39}$$

If $\mathbf{\Sigma}$ is diagonal, this can be rewritten as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{j=1}^{d} \frac{1}{\ell_j^2}(x_j - x'_j)^2\right). \tag{2.40}$$

[5] This term can be a bit misleading though, since every kernel can be defined as a dot product in some feature space.

Moreover, if we have a spherical $\boldsymbol{\Sigma}$ we get an isotropic kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right). \tag{2.41}$$

In these formulations, ℓ is usually referred as a *lengthscale* hyperparameter.

It is possible to prove that the SE kernel is equivalent to $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle$ where $\boldsymbol{\phi}$ is a mapping from $\mathbf{x}$ to an infinitely sized feature space where each point corresponds to a *radial* basis function centered in a value in $\mathbb{R}$ (Rasmussen and Williams, 2006, Chap. 4.2). For this reason, the SE is mostly known as a *radial basis function* (RBF) kernel.

2.2.2 Kernel methods in NLP

Kernel methods were popularised by the advent of Support Vector Machines (SVMs) (Boser et al., 1992; Cortes and Vapnik, 1995). SVMs are similar to linear and logistic regression in formulation but are built on the principles of maximising *margins*, which are regions in the input space which are close to the curve (in regression) or decision boundary (in classification). They also do not easily translate into a probabilistic interpretation since they incorporate sparsity-inducing loss functions directly into training.

In regression, SVMs were largely used in previous work in Quality Estimation, where we aim at measuring the quality of automatically translated texts. Much of this work relies on a small set of hand-crafted features and the corresponding models can benefit from assuming non-linear relationships between these features and the quality labels (Moreau and Vogel, 2014; Specia, 2011; Turchi et al., 2015). In addition, the success of SVMs made them the current choice for the baseline systems provided in the WMT workshop QE shared tasks (Bojar et al., 2013; Callison-Burch et al., 2012, *inter alia).* SVMs were also employed in many instances of text classification (Cancedda et al., 2003; Joachims, 1998; Lodhi et al., 2002; Wang and Manning, 2012, *inter alia)* and in structured prediction (Tsochantaridis et al., 2005).

The ability to easily model non-linearities made kernel methods such as SVMs the state-of-the-art for some NLP tasks during many years. But much like ridge regression, SVMs are only able to provide point estimates when predicting response variables. For classification, a proxy for uncertainty estimates can be obtained from SVMs by measuring the margin between an input and the decision hyperplane. However, there is no guarantee these estimates come from well-calibrated probability distributions and it is not known how to use a similar procedure for regression. Ideally, we would like to combine the representational power of

kernel methods with proper uncertainty modelling using a Bayesian framework. This is the topic of the next Section.

2.3 A Bayesian Kernel Framework: Gaussian Processes

In Section 2.1.1 we mentioned the need for accurate uncertainty estimates and how Bayesian linear regression can help achieve that goal by integrating over the weight parameters. The same reasoning can be applied to kernel-based regression. To do that, let's first restate Equation 2.18 in terms of an arbitrary basis function expansion $\boldsymbol{\phi}$:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|\boldsymbol{\phi}(\mathbf{x}_*), \mathbf{w}) p(\mathbf{w}|\boldsymbol{\Phi}(\mathbf{X}), \mathbf{y}) d\mathbf{w}, \tag{2.42}$$

$$= \int \mathcal{N}(y_*|\boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{w}, \sigma^2)\mathcal{N}(\mathbf{w}|\mathbf{w}_n, \mathbf{V}_n) d\mathbf{w}, \tag{2.43}$$

$$= \mathcal{N}(y_*|\boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{w}_n, \sigma_n^2(\boldsymbol{\phi}(\mathbf{x}_*))), \tag{2.44}$$

$$\mathbf{w}_n = \frac{1}{\sigma^2}\mathbf{V}_n\boldsymbol{\Phi}(\mathbf{X})^T\mathbf{y}, \tag{2.45}$$

$$\mathbf{V}_n = \sigma^2\left(\frac{\sigma^2}{\tau^2}\mathbf{I}_n + \boldsymbol{\Phi}(\mathbf{X})^T\boldsymbol{\Phi}(\mathbf{X})\right)^{-1}, \tag{2.46}$$

$$\sigma_n^2(\boldsymbol{\phi}(\mathbf{x}_*)) = \sigma^2 + \boldsymbol{\phi}(\mathbf{x}_*)^T\mathbf{V}_n\boldsymbol{\phi}(\mathbf{x}_*). \tag{2.47}$$

where $\boldsymbol{\Phi}$ corresponds to applying $\boldsymbol{\phi}$ to each row of $\mathbf{X}$. If we set $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$ we recover Bayesian linear regression.

We can obtain an equivalent formulation in the dual form. Let $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\mathbf{x}_*)$ and $\boldsymbol{\Phi} = \boldsymbol{\Phi}(\mathbf{X})$. Then we can rewrite Equation 2.42 as:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_*|\mu_*, \sigma_*^2) \tag{2.48}$$

$$\mu_* = \tau^2\boldsymbol{\phi}_*^T\boldsymbol{\Phi}^T(\mathbf{K} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y} \tag{2.49}$$

$$\sigma_*^2 = \tau^2\boldsymbol{\phi}_*^T\boldsymbol{\phi}_* - \tau^2\boldsymbol{\phi}_*^T\boldsymbol{\Phi}^T(\mathbf{K} + \sigma^2\mathbf{I}_n)^{-1}\tau^2\boldsymbol{\Phi}\boldsymbol{\phi}_* \tag{2.50}$$

where $\mathbf{K} = \tau^2\boldsymbol{\Phi}\boldsymbol{\Phi}^T$. The main aspect of this formulation is that expansions always appear in the form of $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$, $\boldsymbol{\phi}_*^T\boldsymbol{\Phi}^T$ or $\boldsymbol{\phi}_*^T\boldsymbol{\phi}_*$, scaled by τ^2. This means we can define a kernel

$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})^T \boldsymbol{\psi}(\mathbf{x}')$, where $\boldsymbol{\psi}(\mathbf{x}) = \tau \boldsymbol{\phi}(\mathbf{x})$ and rewrite Equation 2.48 as

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_*|\mu_*, \sigma_*^2) \tag{2.51}$$

$$\mu_* = \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{y} \tag{2.52}$$

$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{k}_* \tag{2.53}$$

where $\mathbf{K}$ is now the Gram matrix with respect to $k(\mathbf{x}, \mathbf{x}')$.

The result above is of central importance to this thesis since it defines a regression model with a *Gaussian Process* prior over $f = \boldsymbol{\psi}(\mathbf{x})^T \mathbf{w}$. To understand that, we can derive the same model from a GP perspective and obtain the same result for the predictive posterior, which we will do next.

We first define GPs in a formal way. A GP is a distribution over functions:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{2.54}$$

where $m(\mathbf{x})$ is the *mean* function and $k(\mathbf{x}, \mathbf{x}')$ is the kernel or *covariance* function. In this sense, they are analogous to Gaussian distributions, which are also defined in terms of a mean and a variance values, or in the case of multivariate Gaussians, a mean vector and a covariance matrix. In fact, a GP can be interpreted as ∞-dimensional Gaussian distribution.

Assuming a GP prior over f with mean function $\mathbf{0}$, we can have a regression model using a formulation similar to Equation 2.1:

$$f \sim \mathcal{GP}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}')), \tag{2.55}$$

$$y_i = f(\mathbf{x}_i) + \epsilon, \tag{2.56}$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2). \tag{2.57}$$

Here we will employ a fundamental property of GPs: *marginalisation*. For an arbitrary set of points in the function space, a GP reduces to a multivariate Gaussian with *all other points* marginalised out. We use this property to condition f on the set of inputs $\mathbf{x}$, resulting in a latent vector $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$.

At prediction time our goal is to obtain f_* for an unseen input $\mathbf{x}_*$. The joint probability of the observed response variables $\mathbf{y}$ and f_* is

$$\begin{pmatrix} \mathbf{y} \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K} + \sigma^2 \mathbf{I}_n & \mathbf{k}_* \\ \mathbf{k}_*^T & k(\mathbf{x}_*, \mathbf{x}_*) \end{pmatrix}\right) \tag{2.58}$$

Conditioning on $\mathbf{y}$, we obtain the predictive posterior for f_*:

$$p(f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(f_*|\mu_*, \sigma_*^2) \tag{2.59}$$

$$\mu_* = \mathbf{k}_*^T(\mathbf{K} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y} \tag{2.60}$$

$$\sigma_*^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T(\mathbf{K} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{k}_*. \tag{2.61}$$

This is equivalent to Equation 2.51 when we assume $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})^T\boldsymbol{\psi}(\mathbf{x}')$ and $f_* = y_*$, which is true since noise is only assumed for observed variables. Therefore we can see that a kernelised Bayesian regression is equivalent to a GP regression model.

2.3.1 Hyperparameter optimization

Until now we assumed model hyperparameters such as the noise variance σ^2 were given. A fully Bayesian approach would treat them as random variables and put prior distributions over them. Inference is then done by marginalising the hyperparameters but this usually result in intractable integrals. Approximate solutions can be found via sampling (see for instance, Murray and Adams (2010)) but this can quickly become a performance bottleneck.

A more common approach is to find point estimates for hyperparameters through some sort of optimisation. In non-Bayesian models the most common method employs a *validation set*, which is disjoint from the training set. The idea is to train a set of models with different hyperparameter values and select the one which performs best on the validation set, with respect to some error metric. An extension of this approach is *cross-validation*, where the full dataset is split into N different training/validation pairs and the decision is made on the average performance over all pairs.

The procedure above relies on a good way to select the hyperparameter values to be tested. *Grid search* (Bishop, 2006, Chap. 1.3) is a common approach, where values are selected based on a grid over a specified range. More recent approaches include *random search* (Bergstra and Bengio, 2012), which selects values randomly according to some distribution (usually uniform), and *Bayesian optimisation* (Shahriari et al., 2015), which defines probabilistic models over the hyperparameter space and uses uncertainty to decide which regions to explore. These methods work well if the number of hyperparameters is small but they can quickly become unfeasible as we increase this number since it becomes harder to explore the full hyperparameter space. This is particularly problematic in kernel methods where we might want to employ more powerful kernels and/or combinations with a larger number of hyperparameters. For instance, the SE kernel with diagonal covariance

(Equation 2.40) has one hyperparameter per dimension in the input vectors. Optimizing such models using these methods can quickly become impractical.

GPs have an elegant solution for that problem: maximising the marginal likelihood with respect to the full training data. This is referred in the literature as *empirical Bayes* or *evidence procedure*. The main advantage of this method is that we can define gradients of the marginal likelihood and employ gradient ascent optimisers, which are much faster than grid and random search. It also obviates the need of a validation set, making full use of the whole available training data.

The marginal likelihood can be defined as

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \int_f p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, f)df, \tag{2.62}$$

where $\boldsymbol{\theta}$ represents the full set of hyperparameters. Since the integrand is Gaussian we can obtain a closed formula solution for the marginal (log) likelihood:

$$\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{\mathbf{y}^T\bar{\mathbf{K}}^{-1}\mathbf{y}}{2} - \frac{\log|\bar{\mathbf{K}}|}{2} - \frac{n\log 2\pi}{2}, \tag{2.63}$$

where $\bar{\mathbf{K}} = \mathbf{K} + \sigma^2\mathbf{I}_n$.

The three terms in the likelihood formula are interpretable: the first one is the *data-fit* term and it is the only one that depends on the outputs; the second one is the *complexity penalty*, which depends only on the inputs and the third one is a normalisation constant. Intuitively, the optimisation procedure balances between complex models that highly fit the data and simple models that give a lower complexity penalty.

The optimisation step requires taking the likelihood derivatives with respect to the hyperparameters:

$$\frac{\partial \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{2}\mathrm{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \bar{\mathbf{K}}^{-1})\frac{\partial \bar{\mathbf{K}}}{\partial \boldsymbol{\theta}}\right), \tag{2.64}$$

where $\boldsymbol{\alpha} = \bar{\mathbf{K}}^{-1}\mathbf{y}$. The derivatives of $\bar{\mathbf{K}}$ depend on the kernel used in the model. This shows another nice property of this method: any kernel can be optimised this way as long as its gradient vector is defined.

On Figure 2.5 we generate a dataset by drawing random samples and pushing them through a noisy "cubic sine" function $y = (\sin(x) + \epsilon)^3$, where $\epsilon \sim \mathcal{N}(0, 0.2)$, and plot a GP with an SE kernel, before and after hyperparameter optimisation. We can see that this

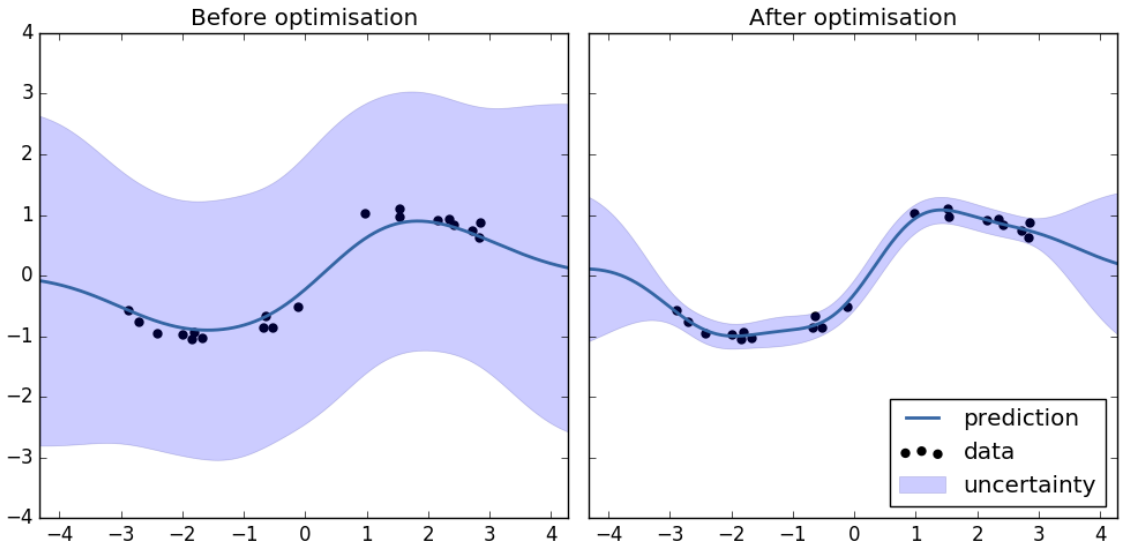


Fig. 2.5 A GPs with 1-D SE kernel fitted to the "cubic sine" data. On the left we show the fit before optimising the hyperparameters, which were all initialised to 1.0. On the right we show the fit after optimising the marginal likelihood.

procedure results in a much better fit. In GP terminology we usually refer to this optimisation step as “training”, even though this is done at the hyperparameter level.

It is important to note that there is no guarantee that the marginal likelihood is convex, so this procedure is prone to local maxima. Random restarts is a common method to avoid bad optima and it is usually effective for the GP models we will address in this thesis.

2.3.2 Automatic relevance determination

In Section 2.2.1 we introduced the SE kernel in various formulations. The isotropic version is the most commonly used in SVMs since it has only one lengthscale hyperparameter. However, we can also have an anisotropic SE kernel, where we have one lengthscale per dimension in the input vectors, as shown in Equation 2.40. While this would be impractical in SVMs and other kernel machines, in GPs we can use empirical Bayes to efficiently optimise the lengthscales. This procedure is called *automatic relevance determination* (ARD) and it is commonly employed in the GP literature.

ARD can be interpreted as performing feature selection. Low lengthscale values correspond to dimensions where the corresponding difference in the values of $\mathbf{x}$ and $\mathbf{x}'$ are high and therefore discriminative. Conversely, high lengthscale values result in low values in the corresponding dimension, indicating that the feature does not contribute as much for the

model. ARD is very useful when models are based on hand-crafted feature templates not only because they can help prediction but are also interpretable.

2.3.3 Matèrn kernels

Previous work in the SVM literature mostly focuses on linear, polynomial and SE kernels. There is however a myriad of other kernels with a varied set of properties which can be useful when there is some intuition about the behaviour of the function we are trying to model. Here we introduce the Matèrn class of kernels (Rasmussen and Williams, 2006, Sec. 4.2.1), which is widely used in the GP literature for modelling noisy behaviour.

The SE kernel assumes strong smoothness of the corresponding GP samples. This is because a GP with an SE kernel is infinitely mean-square differentiable.[6]. A Matèrn kernel of order ν relaxes this assumption by encoding functions ν-times mean-square differentiable only. For $\nu \to \infty$ it reduces to the SE kernel, so it can be seen as a generalisation of the latter.

Matèrn kernels have a rather involved computation which can be simplified when ν is a half-integer. Most used values for ν are $3/2$ and $5/2$, giving raise to the Matèrn $3/2$ kernel

$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{\ell}\right) \exp\left(-\frac{\sqrt{3}|\mathbf{x} - \mathbf{x}'|}{\ell}\right) \tag{2.65}$$

and the Matèrn $5/2$ kernel

$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{\ell} + \frac{5(\mathbf{x} - \mathbf{x}')^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{\ell}\right). \tag{2.66}$$

For values of ν higher than $5/2$ it becomes harder to differentiate between the corresponding Matèrn kernel and an SE kernel (Rasmussen and Williams, 2006, Sec. 4.2.1).

Notice that Matèrn kernels also have a lengthscale hyperparameter, like SE kernels. This means we can also extend them to employ feature selection via ARD, by having one lengthscale per input dimension.

[6]Mean-square differentiability is a stochastic calculus concept which can loosely be interpreted as differentiability in deterministic calculus (Rasmussen and Williams, 2006, Sec. 4.1.1).

2.3.4 Bias and noise kernels

Bias and noise kernels are two common classes of kernels that are also widely used in GPs. A bias kernel is defined as simply outputting a constant value regardless of the inputs,

$$k(\mathbf{x}, \mathbf{x}') = b \tag{2.67}$$

where b is the bias hyperparameter. While it is not very useful on its own, it can be used in combination with other kernels in order to encode different aspects of the function we are interested in modelling. We will see a few examples in Section 2.3.5.

A noise kernel is similar to the bias kernel, but only outputs its value if $\mathbf{x}$ and $\mathbf{x}'$ are the same,

$$k(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \mathbb{I}(\mathbf{x}, \mathbf{x}') \tag{2.68}$$

where σ_n^2 is the noise hyperparameter and $\mathbb{I}$ is an indicator function that returns 1 if the arguments are the same and 0 otherwise. This kernel returns a diagonal Gram matrix, with diagonal values equal to σ_n^2. It is equivalent to the Gaussian likelihood term in the GP regression model. This allows one to encode the likelihood noise as a kernel hyperparameter, which is useful in practice since it can ease implementation efforts.

2.3.5 Combining kernels

Kernels have a set of attractive properties that allow the practitioner to combine different kernels when building a GP model. Here we list some of these properties:

Sum of kernels If $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ are kernels then $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$ is a kernel.

Product of kernels If $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ are kernels then $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel.

Product by a function If $k(\mathbf{x}, \mathbf{x}')$ is a kernel and $g(\mathbf{x})$ is a deterministic function then $g(\mathbf{x})k(\mathbf{x}, \mathbf{x}')g(\mathbf{x}')$ is a kernel.

Normalisation If $k(\mathbf{x}, \mathbf{x}')$ is a kernel then

$$\hat{k}(\mathbf{x}, \mathbf{x}') = \frac{k(\mathbf{x}, \mathbf{x}')}{\sqrt{k(\mathbf{x}, \mathbf{x})k(\mathbf{x}', \mathbf{x}')}} \tag{2.69}$$

is a kernel. This can be proved by the using the product by a function property and let $g(\mathbf{x}) = k(\mathbf{x}, \mathbf{x})^{-1/2}$. The normalised kernel ensures the output is always between 0 and 1.

Direct sum If $k_1(\mathbf{x}_1, \mathbf{x}'_1)$ and $k_2(\mathbf{x}_2, \mathbf{x}'_2)$ are kernels defined over different spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ then the direct sum $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}_1, \mathbf{x}'_1) \oplus k_2(\mathbf{x}_2, \mathbf{x}'_2)$ is a kernel.

Kronecker product If $k_1(\mathbf{x}_1, \mathbf{x}'_1)$ and $k_2(\mathbf{x}_2, \mathbf{x}'_2)$ are kernels defined over different spaces $\mathcal{X}_1$ and $\mathcal{X}_2$ then the Kronecker (or tensor) product $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}_1, \mathbf{x}'_1) \otimes k_2(\mathbf{x}_2, \mathbf{x}'_2)$ is a kernel.

The first two properties are particularly useful when employed with the bias kernel introduced in Section 2.3.4. Adding a bias term b to a GP kernel is useful when we expect the range of the response variables to not be centered around zero. In fact, this is equivalent to defining a GP model where the mean function is a constant function with value b instead of 0. Multiplying a stationary kernel such as SE or Matèrn by a bias term greater than 0 introduces a variance term which encodes variations in the response variable, similar to how the lengthscale hyperparameters encode variations in the inputs. We will revisit these and other properties throughout this thesis when applying GPs in real world tasks.

2.3.6 Gaussian Processes in NLP

Even though Gaussian Processes have been studied in ML for more than 20 years, only very recently they were applied in NLP problems. Cohn and Specia (2013) employed a multi-task version of a GP for Quality Estimation data where multiple annotations per sentence were available. Their results show improved performance over single task baselines and a SVM regression model using EasyAdapt (Daumé III, 2007), a technique for domain adaptation. In fact, they show that a multi-task GP generalises the EasyAdapt model. Shah et al. (2013) also used GPs for QE but they focused on feature analysis instead. They employ ARD kernels and use the optimised hyperparameters to find which features are more discriminative for the task.

Another body of work uses GPs in non-regression scenarios. Polajnar et al. (2011) applied GPs for text classification in a biomedical environment. Classification can be done using GPs by replacing the Gaussian likelihood with a Bernoulli or Categorical one. This makes the model intractable so it needs to be approximated through techniques such as Laplace approximation (Williams and Barber, 1998) or Expectation Propagation (Minka, 2001). Preoţiuc-Pietro and Cohn (2013) used GPs for modelling word periodicities in social

media data. They show how different kernels can be applied to hashtags in Twitter and propose new variants which are better at modelling periodic behaviour. Lukasik et al. (2015) employed point processes based on GPs to model rumour dynamics in social media. Their proposed model outperforms baselines when predicting rates of rumour spreading across time. Finally, GPs were also combined with CRFs for structured prediction problems (Altun et al., 2004; Bratières et al., 2013).

GPs are a flexible and powerful framework for modelling non-linear behaviour in the data while taking uncertainty into account. This makes them very suitable for our goal of modelling response variable confidence in Text Regression. In the next Section we show how to evaluate uncertainty estimates in regression.

2.4 Evaluation

The models studied until now are able to predict response variables for unseen inputs in the form of probability distributions. Evaluating predictive performance is usually done in terms of metrics with respect to a set of labelled unseen inputs or *test set*. Since we are interested in assessing our models in terms of their full predictive distributions we need metrics that can take those into account.

A simple and effective way to take distributions into account is to calculate the likelihood of the gold standard values with respect to these distributions. This was proposed by Quiñonero-Candela et al. (2006) under the name of *Negative Log Predictive Density* (NLPD), which is defined as

$$\mathrm{NLPD}(p(\hat{\mathbf{y}}), \mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n} \log p(\hat{y}_i = y_i | \mathbf{x}_i). \tag{2.70}$$

Since this is an error metric, lower values are better. Intuitively, if two models produce equally incorrect predictions but they have different uncertainty estimates, NLPD will penalise the overconfident model more than the underconfident one. Conversely, if predictions are close to the true value then NLPD will penalise the underconfident model instead.

On Figure 2.6 we plot the NLPD for a Gaussian predictive distribution with different variance values and fixed $(y - \mu)^2 = 1$. We can see that it favours conservative models, that is, models that are underconfident in their predictions.

NLPD is widely used in the GP literature when evaluating regression models, especially due to its simplicity. This will be the our metric of choice when evaluating the models proposed in this thesis with respect to their predictive performance.

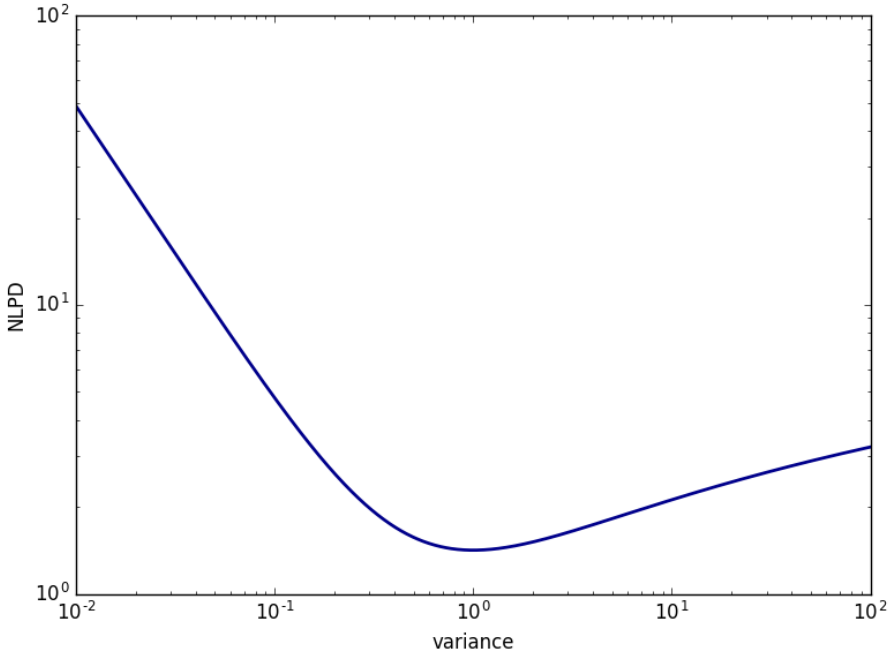


Fig. 2.6 NLPD values for a Gaussian predictive distribution with different variance values (in logarithmic scale). Based on Figure 7 in Quiñonero-Candela et al. (2006).

2.4.1 Decision theory for regression

When employing regression models in a decision making scenario it might be the case we are interested in single point estimates from such distributions. To obtain these values we need to know which *loss function* we are trying to minimize and then derive the so-called *Bayes estimator*.

Many common loss functions used in regression are based on aggregate error measurements with respect to the gold standard values. The *Mean Absolute Error* (MAE) is such a metric:

$$L_{\text{MAE}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} |\hat{y}_i - y_i|, \tag{2.71}$$

where $\hat{\mathbf{y}}$ are the model predictions and $\mathbf{y}$ are the gold standard values. Another example is the *Root Mean Squared Error* (RMSE):

$$L_{\text{RMSE}}(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}. \tag{2.72}$$

This loss assigns a quadratic penalty to errors so, compared to MAE, it is more sensitive to wrong predictions but also to outliers.

It is possible to prove that the Bayes estimators for the MAE and the RMSE are the median and the mean of the predictive distribution (Murphy, 2012, Sec. 5.7). In standard GPs (including Bayesian linear regression) predictions are Gaussians, where the mean and median coincide. So, to minimize either MAE or RMSE it is enough to consider the mean as the optimal estimator. This is also the case of ridge regression, since predictions are Gaussians but with fixed variance for every input.

An alternative way to assess the predictive performance is by measuring the correlation between the full set of predictions and the gold standard, using the Pearson's r correlation metric, for instance:

$$r(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\sum_{i=1}^{n}(\hat{y}_i - \hat{\bar{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \hat{\bar{y}})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{2.73}$$

where $\bar{y}$ and $\hat{\bar{y}}$ are the empirical means with respect to the gold standard and the predictions.

It is not known if there are efficient Bayes estimators for Pearson's r. The standard practice is to calculate this metric with respect to the mean or the median of the distributions.

2.5 Discussion

In this Chapter we introduced Gaussian Processes for regression. We show how they can model complex, non-linear behaviour in the data while being able to incorporate uncertainty and provide probability distributions as predictions. We also discussed how to intrinsic evaluate these models via NLPD and discussed aspects of decision theory when it is desirable to have point estimate predictions.

A key aspect of GPs is their ability to encode information in the dataset via kernels. We showed in this Chapter how such kernels can be useful to model different function behaviours as well as other settings. In fact, a kernel can be interpreted as a similarity function between two datapoints. In the next Chapter we are going to show how to harness this concept to build powerful text representations which go beyond standard ones used in the NLP literature.

Chapter 3

Structural Kernels

In the previous Chapter we introduced GPs and showed how training and prediction can be framed in terms of kernels between inputs. A kernel can be interpreted as a similarity metric between two instances, without requiring an explicit representation in terms of features. This property allows the use of complex input representations and reduces the burden on feature engineering. In this Chapter we explore this property to define *structural kernels*, which aim at measuring the similarity between discrete structures.

The concept of structural kernels was popularised by the seminal work of Haussler (1999), who introduced *R-convolution* kernels. The intuition behind them is to measure the similarity between two discrete objects by dividing them into smaller substructures (as defined by a relation R) and measuring how many of these substructures are shared between the two objects. In that sense an instance of an R-convolution kernel can be seen as defined by the type of structure and how we decide to segment it into smaller substructures. For texts, a simple example would be to split it at the word level and count how many words two different texts have in common, which would reduce to a bag-of-words representation.

The main power of such kernels arise when the structure decomposition results in an exponentially-sized representation. This is because it is possible to define efficient algorithms based on dynamic programming to obtain the kernel values in that space implicitly. In this Chapter we focus on two classes of structures, strings and trees. Kernels for these structures were developed in past work and successfully employed in a number of NLP tasks.

A key question that arises when dealing with structural kernels is how to set their hyperparameters. Most previous work either set these to default values or use simple techniques like grid search. As discussed in Section 2.3.1, employing kernels inside GPs gives the benefit of having efficient model selection procedures via empirical Bayes. Therefore, a natural question is if we can employ such techniques when using structural kernels as

well. In this Chapter, we investigate this question by: 1) extending string and tree kernels with hyperparameter gradient derivations and 2) proposing new parameterisations which enhance their model capacity. Heavily parameterised kernels are difficult to optimise using simple procedures like grid or random search but are much more feasible in a GP empirical Bayes framework. We validate this proposed approach by conducting a series of benchmarks aiming at checking how much data is needed to learn kernel hyperparameters.

Section 3.1 introduces string kernels, which can measure the similarity between two strings in terms of their substrings, potentially including gaps between symbols. In Section 3.2 we propose a new definition for these kernels using vector and matrix multiplications, an approach we call Vectorised String Kernels and we derive their gradients with respect to their hyperparameters. Finally, in Section 3.3 we introduce tree kernels and propose a new generalisation based on symbol specific hyperparameters, while also deriving their gradients.

The string kernel explanations in Section 3.1 are based on Lodhi et al. (2002) and Cancedda et al. (2003), while the original tree kernel derivations in Section 3.3 are inspired by Collins and Duffy (2001) and Moschitti (2006b). Sections 3.2, 3.3.1 and 3.3.2 are completely novel.

3.1 String Kernels

Strings are the most straightforward way to represent natural language so using string kernels is a natural first choice if we want to use structural kernels to model our data. The main idea is to compare two strings in terms of how many substrings they share. A crucial choice when defining which substrings to consider is to enforce them to be *contiguous* or not. Kernels can be defined for either scenario but for the former it is also possible to employ an explicit representation, since the total number of features is quadratic in the string size. In this setting, where we can efficiently store the underlying feature vectors, it is usually better to skip kernelisation and devise algorithms to work directly on explicit representations.

The main power of string kernels arises when we want to consider *non-contiguous* substrings as a representation signal. In this setting, explicit enumeration is unfeasible because the feature space becomes exponential in string size. However, string kernels can deal with such spaces by using the kernel trick through efficient dynamic programming methods.

To define string kernels formally we are going to use a notation similar to Lodhi et al. (2002) and Cancedda et al. (2003):

- An *alphabet* Σ is a set of symbols. $|\Sigma|$ denotes its size.

- A *string* $s = s_1, \ldots, s_{|s|}$ is a sequence of symbols from Σ, where $|s|$ is the length of the sequence. The empty string is denoted by ϵ.
- The set of all strings with length n is denoted by Σ^n and the set of all finite strings is Σ^*:

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \tag{3.1}$$

- A concatenation of two strings s and s' is denoted by ss'. Likewise, sa is the concatenation of a string s with the symbol a.
- A substring $s[i : j]$ is the substring $s_i \ldots s_j$ of s. For notation simplicity, we also define $s[: j] = s[1 : j]$, the prefix of s up to symbol s_j, and $s[: -1] = s[1 : |s| - 1]$, the substring corresponding to all symbols in s except for the last one.
- We define u as a subsequence of s if there exist indices $\mathbf{i} = (i_1, \ldots, i_{|u|})$ with $1 \leq i_1 < \cdots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$ for $j = 1, \ldots, |u|$ or $u = s[\mathbf{i}]$ for short. Intuitively, u can be interpreted as substring of s while allowing *gaps* between symbols.
- The length or *span* of a subsequence or $u = s[\mathbf{i}]$ is defined as $\ell(\mathbf{i}) = i_{|u|} - i_1 + 1$.

A string kernel of length n between two strings s and s' is defined as

$$k_n(s, s') = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}: s[\mathbf{i}]=u} \sum_{\mathbf{j}: s'[\mathbf{j}]=u} \lambda^{\ell(\mathbf{i})+\ell(\mathbf{j})}, \tag{3.2}$$

where $\lambda \in (0, 1]$ is a *decay* hyperparameter that penalises non-contiguous subsequences. Intuitively, this kernel matches all subsequences of length n, including non-contiguous ones, with each occurrence being discounted according to their overall span. Having $\lambda < 1$ is usually necessary because otherwise the Gram matrix becomes diagonal since two equal strings will share, on average, exponentially more subsequences compared to two different ones.

A naive computation of this kernel enumerating all possible subsequences would require $O(|\Sigma^n|)$ time and space. However, Lodhi et al. (2002) devised a recursive formulation for this kernel, leading to an efficient dynamic programming solution which reduces complexity

to $O(n|s||s'|)$:

$$k'_0(s,s') = 1, \text{for all } s, s', \tag{3.3}$$

$$\text{for all } i = 1,\ldots,n-1:$$

$$k''_i(s,s') = 0, \text{if } \min(|s|,|s'|) < i \tag{3.4}$$

$$k'_i(s,s') = 0, \text{if } \min(|s|,|s'|) < i \tag{3.5}$$

$$k''_i(sa,s'b) = \lambda k''_i(sa,s') \text{ iff } b \neq a \tag{3.6}$$

$$k''_i(sa,s'a) = \lambda k''_i(sa,s') + \lambda^2 k'_{i-1}(s,s') \text{ otherwise} \tag{3.7}$$

$$k'_i(sa,s') = \lambda k'_i(s,s') + k''_i(sa,s') \tag{3.8}$$

$$\text{and finally:}$$

$$k_n(s,s') = 0, \text{if } \min(|s|,|s'|) < n \tag{3.9}$$

$$k_n(sa,s') = k_n(s,s') + \sum_{j:s'_j=a} \lambda^2 k'_{n-1}(s, s'[:j-1]) \tag{3.10}$$

We now show how to reach this derivation in detail, following the explanation in Cancedda et al. (2003). Assume $\lambda = 1$, for simplicity. Suppose we already know the value for $k_n(s,s')$, what do we need to obtain $k_n(sa,s')$, for a symbol $a \in \Sigma$? The recursion relies on two properties:

1. All subsequences common to s and s' are also common to sa and s';
2. We also need to consider new matching subsequences ending in a occurring in s' and which prefix (of length $n-1$) occur in s. Notice that the prefix can be non-contiguous.

These properties mean we can reuse the value for $k_n(s,s')$ and just add the new matches arising from appending the symbol a to s. This translates to the following

$$k_n(sa,s') = k_n(s,s') + \sum_{j:s'_j=a} \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{l}:s'[\mathbf{l}]=u, l_{n-1}<j} 1. \tag{3.11}$$

Let's describe each summation in the second term. The first one scans all occurrences of a in s', since we need to check for all possible new matches ending in a. The second summation iterates over all (possibly non-contiguous) prefixes of length $n-1$. Intuitively, these will be "joined" with a to form new matches. The third summation checks for every occurrence of the prefix u in s. Since we are actually calculating the kernel over sa, this term effectively check for all new subsequences ending with the new symbol a. Finally, the fourth summation checks for occurrences of u in s' *as long as their appear before* a. This last condition is

represented by the check $l_{n-1} < j$, where j is the index of the symbol a currently being considered in the first summation. Notice that if s' has multiple occurrences of a, some prefixes might be counted multiple times depending on their position in s'. This is expected, since we need to account for *all* new matches ending in a.

If we analyse Equation 3.11 we can see that the last three summations are similar to Equation 3.2, but for subsequence length $n-1$ and with the condition on the positions of a in s'. Therefore we can rewrite Equation 3.11 as

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} k_{n-1}(s, s'[:j-1]). \tag{3.12}$$

The reason why this works is because we are calling k_n using s and *substrings of* s' as arguments. These substrings will always end in a symbol right before an occurrence of a in s', therefore there is no need for the condition $l_{n-1} < j$ anymore. The recursion finishes at the base cases

$$k_0(s, s') = 1, \text{for all } s, s', \tag{3.13}$$

$$k_n(s, s') = 0, \text{if } \min(|s|, |s'|) < n. \tag{3.14}$$

Intuitively, the algorithm iterates over all prefixes in s, starting with a single symbol and ending with the whole string. At each iteration, it compares with the whole s', counting and matching subsequences and reusing these values in subsequent iterations over s.

The overall complexity for this formulation is $O(n|s||s'|^2)$. The summation in Equation 3.12 has $O(|s'|)$ terms, each one of them extending to other $O(|s'|)$ terms n times, bringing the complexity for the summation to $O(n|s'|^2)$. Since Equation 3.12 iterates over all symbols in s, this brings us to the final complexity.

Let's now extend this reasoning for general values of λ. Recall from Equation 3.2 that each subsequence match is discounted by $\lambda^{\ell(\mathbf{i})}\lambda^{\ell(\mathbf{j})} = \lambda^{\ell(\mathbf{i})+\ell(\mathbf{j})}$, where $\ell(\mathbf{i})$ and $\ell(\mathbf{j})$ are the lengths of the subsequence spans in s and s', respectively. Therefore, new matches have to be discounted according to their spans as well. This translates to this generalisation of Equation 3.11:

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{l}:s'[\mathbf{l}]=u, l_{n-1}<j} \lambda^{|s|+1-i_1+1}\lambda^{j-l_1+1} \tag{3.15}$$

$$= k_n(s, s') + \sum_{j:s'_j=a} \lambda^2 \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{l}:s'[\mathbf{l}]=u, l_{n-1}<j} \lambda^{|s|-i_1+1}\lambda^{j-1-l_1+1}. \tag{3.16}$$

The two discount terms in the first row correspond to the lengths of the corresponding matched subsequence spans. For s, this is $|s| + 1 - i_1 + 1$, since the span starts in i_1 and always end with the last symbol (a). For s', this is $j - l_1 + 1$ because the span starts in l_1 and ends with the occurrence of a in s' being considered in the loop. Now defining

$$k'_{n-1}(s, s') = \sum_{u \in \Sigma^{n-1}} \sum_{\mathbf{i}: s[\mathbf{i}]=u} \sum_{\mathbf{j}: s'[\mathbf{j}]=u} \lambda^{|s|-i_1+1} \lambda^{|s'|-j_1+1}, \tag{3.17}$$

we can rewrite Equation 3.16 as

$$k_n(sa, s') = k_n(s, s') + \sum_{j: s'_j=a} \lambda^2 k'_{n-1}(s, s'[: j-1]). \tag{3.18}$$

Unlike the case where $\lambda = 1$, here we have to define a new function k'_{n-1} to enable the recursion because the discount terms are different from k_{n-1}. Intuitively k'_{n-1} counts matching subsequences of $n-1$ symbols but discount them according to the span covering the first symbol in the subsequence *until the end of the string*. The idea is to store "masses" of matches of length $n-1$ ready to be turned into matches of length n should the next symbol in s match some of the symbols in s'. The values of k'_n can be calculated recursively as

$$\begin{aligned}
k'_n(sa, s') = & \sum_{u \in \Sigma^{n-1}} \sum_{\mathbf{i}: s[\mathbf{i}]=u} \sum_{\mathbf{j}: s'[\mathbf{j}]=u} \lambda^{|s|+1-i_1+1} \lambda^{|s'|-j_1+1} \\
& + \sum_{j: s'_j=a} \sum_{v \in \Sigma^{n-1}} \sum_{\mathbf{i}: s[\mathbf{i}]=v} \sum_{\mathbf{l}: s'[\mathbf{l}]=v, l_{n-1}<j} \lambda^{|s|+1-i_1+1} \lambda^{|s'|-l_1+1} && (3.19) \\
= & \lambda \sum_{u \in \Sigma^{n-1}} \sum_{\mathbf{i}: s[\mathbf{i}]=u} \sum_{\mathbf{j}: s'[\mathbf{j}]=u} \lambda^{|s|-i_1+1} \lambda^{|s'|-j_1+1} \\
& + \sum_{j: s'_j=a} \sum_{v \in \Sigma^{n-1}} \sum_{\mathbf{i}: s[\mathbf{i}]=v} \sum_{\mathbf{l}: s'[\mathbf{l}]=v, l_{n-1}<j} \lambda^{|s|-i_1+1} \lambda^{j-1-l_1+1} \lambda^{|s'|-j+2} && (3.20) \\
= & \lambda k'_n(s, s') + \sum_{j: s'_j=a} k'_{n-1}(s, s'[: j-1]) \lambda^{|s'|-j+2}. && (3.21)
\end{aligned}$$

Summarising, the full recursive formulation can be written as

$$k'_0(s, s') = 1, \text{for all } s, s', \tag{3.22}$$

$$\text{for all } i = 1, \ldots, n-1 :$$

$$k'_i(s, s') = 0, \text{if } \min(|s|, |s'|) < i \tag{3.23}$$

$$k'_i(sa, s') = \lambda k'_i(s, s') + \sum_{j:s'_j=a} k'_{n-1}(s, s'[: j-1])\lambda^{|s'|-j+2} \tag{3.24}$$

$$\text{and finally:}$$

$$k_n(s, s') = 0, \text{if } \min(|s|, |s'|) < n \tag{3.25}$$

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} \lambda^2 k'_{n-1}(s, s'[: j-1]) \tag{3.26}$$

This generalisation also has complexity $O(n|s||s'|^2)$. The final kernel formulation reduces complexity by defining the additional function

$$k''_n(sa, s') = \sum_{j:s'_j=a} k'_{n-1}(s, s'[: j-1])\lambda^{|s'|-j+2}. \tag{3.27}$$

Remember that k'_i discount matches by their spans until the end of the string. Intuitively, k''_n stores intermediate values for this and applying a discount corresponding to the remaining symbols in s' (represented by the term $|s'| - j + 2$). This function also admits a recursion, namely,

$$\begin{aligned} k''_n(sa, s'b) &= \sum_{j:s'_j=a} k'_{n-1}(s, s'[: j-1])\lambda^{|s'|+1-j+2} \\ &= \lambda k''_n(sa, s'), \end{aligned} \tag{3.28}$$

if $a \neq b$ and

$$\begin{aligned} k''_n(sa, s'a) &= \sum_{j:s'_j=a} k'_{n-1}(s, s'[: j-1])\lambda^{|s'|+1-j+2} \\ &\quad + k'_{n-1}(s, s')\lambda^{(|s'|+1)-(|s'|+1)+2} \\ &= \lambda k''_n(sa, s') + \lambda^2 k'_{n-1}(s, s'), \end{aligned} \tag{3.29}$$

otherwise. With this, k' can be represented as a function of k'':

$$k'_n(sa, s') = \lambda k'_n(s, s') + k''_n(sa, s'). \tag{3.30}$$

Plugging this in Equation 3.24 we retrieve the original dynamic programming formulation stated in the Equations 3.3 to 3.10. This brings the final complexity to $O(n|s||s'|)$.

The procedure above also produces results for lengths smaller than n as a byproduct, which allows us to define another kernel composed of a linear combination of $k_i, 1 \leq i \leq n$:

$$\bar{k}_n(s, s') = \sum_{i=1}^{n} \mu_i k_i(s, s') \tag{3.31}$$

where μ_i corresponds to the weight given to k_i. This means we can define a kernel which takes into account all lengths up to n without changing the time complexity.

This general formulation can be applied to any alphabet Σ and this choice leads to different kernels. If we consider Σ to be the set of all characters we reach the original definition of Lodhi et al. (2002). However, we can also define the alphabet as the vocabulary of a natural language, where each symbol is represented as a word and symbols in a string are delimited by whitespaces. This leads to *Word-Sequence Kernels* (WSK) (Cancedda et al., 2003), which are string kernels that operate at the word level.

String kernels can be extended in a number of ways. We will discuss some of these extensions in the next Sections.

3.1.1 Independent decay factors for gaps and matches

The main intuition around the decay hyperparameter λ is to decrease the contribution of large span subsequences. In the formulation, this discount occurs even when subsequences are contiguous, i.e., symbol matches are also penalised. To be able to better discriminate the contribution of contiguous and non-contiguous subsequences we can extend the kernel to have two separate decay hyperparameters λ_g and λ_m for gaps and matches (Cancedda et al.,

2003). This can be implemented by replacing Equations 3.6, 3.7, 3.8, and 3.10 by

$$k''_i(sa, s'b) = \lambda_g k''_i(sa, s') \text{ iff } b \neq a, \tag{3.32}$$

$$k''_i(sa, s'b) = \lambda_g k''_i(sa, s') + \lambda_m^2 k'_{i-1}(s, s'), \tag{3.33}$$

$$k'_i(sa, s') = \lambda_g k'_i(s, s') + k''_i(sa, s'), \tag{3.34}$$

$$k_n(sa, s') = k_n(s, s') + \sum_{j:s'_j=a} \lambda_m^2 k'_{n-1}(s, t[: j-1]), \tag{3.35}$$

respectively.

Allowing different decays for gaps and matches gives better flexibility when defining the underlying feature space encoded by the kernel. For instance, having $\lambda_g = 0$ effectively nullifies contributions from all non-contiguous subsequences while setting $\lambda_g = 1$ reduces to a kernel that treats all subsequences with the same symbols equally regardless of contiguity. Notice that this formulation does not change the complexity of the kernel.

3.1.2 Soft matching

The standard definition of the string kernel relies only on exact match symbols (Equations 3.6, 3.7 and 3.10). For words this is undesirable since synonyms or words with related meanings will never be considered similar. This can be avoided by employing soft matching between words, i.e., allowing two words which are similar but not equal to contribute to the kernel calculation (Cancedda et al., 2003).

Formally, soft matching is implemented by defining a *similarity matrix* $\mathbf{S} \in (\mathbb{R}_0^+)^{|\Sigma_*^n| \times |\Sigma_*^n|}$, where $\Sigma_*^n = \Sigma^0 \cup \cdots \cup \Sigma^n$. In other words, $\mathbf{S}$ encodes the similarity values between word sequences. For the kernel to be still valid, this matrix must be PSD[1] (it is symmetric by definition). This can be ensured by having a similarity measure $\mathrm{sim}(a, b) = \mathbf{S}_{a,b}$ between word sequences which is also PSD. If we set sim as a delta function $\mathrm{sim}(a, b) = 1$ if $a = b$ and 0 if $a \neq b$ then we have $\mathbf{S} = \mathbf{I}_{|\Sigma_*^n|}$ and we recover the hard matching approach.

To keep the kernel calculation implicit, it is useful to define $\mathbf{S}_{s,s'} = \prod_{i=1}^{n} \mathrm{sim}(s_i, s'_i)$. In other words, the similarity between two word sequences with equal length is the product of the similarities between individual words contained in the sequences which have the same index. With this assumption we can skip having to store $\mathbf{S}$ explicitly and keep the original

[1]Recall from Section 2.2 that a $d \times d$ matrix $\mathbf{M}$ is positive semi-definite (PSD) if it is symmetric and $\mathbf{x}\mathbf{M}\mathbf{x}^T \geq 0$ for all $\mathbf{x} \in \mathbb{R}^d$.

formulation by replacing Equations 3.6, 3.7 and 3.10 with

$$k''_i(sa, s'b) = \lambda k''_i(sa, s') + \lambda^2 \mathrm{sim}(a, b) k'_{i-1}(s, s') \tag{3.36}$$

$$k_n(sa, s') = k_n(s, s') + \sum_{j}^{|s'|} \lambda^2 \mathrm{sim}(a, s'_j) k'_{n-1}(s, s'[: j-1]) \tag{3.37}$$

To ensure sim is a PSD function we can define it as another kernel. A straightforward way to do this is to represent each word as vector in $\mathbb{R}^d$ and then apply any known kernel, such as a linear dot product. These vectors are usually referred as *word embeddings* in the literature. A simple embedding is the so-called *one-hot encoding*, where $d = |\Sigma|$ and each word is represented as a unit vector in that space, with 1 in the dimension corresponding to the word identifier and 0 in all the others. If we use one-hot encodings and $\mathbf{S}$ is defined as a dot product we again recover hard matching. To obtain true soft matching we need to employ more involved embeddings.

Word embeddings are a heavily studied subject in NLP (Deerwester et al., 1990; Sahlgren, 2006; Turian et al., 2010, *inter alia)*. Most of these studies rely on a definition of context and build on the idea that similar words appear in similar contexts, a property known as *distributional hypothesis* (Harris, 1954). Early work assumed the context to be a whole document and built embeddings based on word co-occurrences inside a single document, a framework known as Latent Semantic Analysis (LSA) (Deerwester et al., 1990). More recent studies define more local contexts such as a window of n-grams around the word of interest (Mikolov et al., 2013; Pennington et al., 2014) or using syntactic dependencies obtained from a parser (Levy and Goldberg, 2014).

A useful common property of these methods is their ability to be trained off-line on large corpora, in an unsupervised manner. This allows using these embeddings as an off-the-shelf resource in NLP tasks. One the other hand there is no consensus about which of them work best, especially because intrinsic evaluation is not trivial. In this thesis we will employ pre-trained GloVe embeddings (Pennington et al., 2014) for the purposes of soft matching. They have achieved state-of-the-art results in a variety of word similarity and analogy tasks and are freely available as a resource.[2]

[2]http://nlp.stanford.edu/projects/glove

3.1.3 A unified algorithm

Algorithm 1 shows the pseudocode for calculating the kernel between two strings, including the extensions shown on Sections 3.1.1 and 3.1.2. The original string kernel can be retrieved by tying the two decay factors into a single one and use one-hot embeddings for each symbol in the matrix $\mathbf{E}$ (to simulate hard matching). Here we use bracket notation to index matrices and tensors (similar to Python syntax). Notice we assume $\mathbf{E}$ can be indexed by a symbol: this can be implemented by assigning a unique integer id for each symbol in the alphabet.

The core calculation happens in the loop at lines $2-7$, where we obtain values for k', represented as a 3d tensor. Each layer in k' is a matrix k'_n of size $|s| \times |s'|$ where each $[i, j]$ element contains common n-length subsequence counts (weighted by the decay hyperparameters) up to the corresponding i and j symbols in s and s'. The final loop at lines $9-14$ inspects the k' tensor and sums the corresponding values where symbols match (again, weighted by the decay hyperparameters). Each k_i contains the kernel result for subsequences of length i and the final result is obtained by a weighted average of each k_i.

Algorithm 1 String kernel

Require: strings s and s', gap decay λ_g, match decay λ_m, ngram order n, ngram order coefficients $\boldsymbol{\mu}$, embeddings matrix $\mathbf{E}$

1: $k' \leftarrow \mathbf{1}$ ▷ 3d tensor with shape $n \times |s| \times |s'|$, initialised as in Equation 3.3
2: **for** $i \leftarrow 0, n-1$ **do** ▷ Fill k' with intermediate kernel values
3: **for** $j \leftarrow 0, |s|-2$ **do**
4: $k'' \leftarrow 0$
5: **for** $k \leftarrow 0, |s'|-2$ **do**
6: $k'' \leftarrow \lambda_g k'' + \lambda_m^2 \mathbf{E}[s_j]\mathbf{E}[s'_k]k'[i, j, k]$ ▷ Equation 3.36
7: $k'[i+1, j+1, k+1] \leftarrow \lambda_g k'[i+1, j, k+1] + k''$ ▷ Equation 3.34
8: $k \leftarrow 0$
9: **for** $i \leftarrow 0, n-1$ **do** ▷ Get final values for each k_n
10: $k_i \leftarrow 0$
11: **for** $j \leftarrow 0, |s|-1$ **do**
12: **for** $k \leftarrow 0, |s'|-1$ **do**
13: $k_i \leftarrow k_i + \lambda_m^2 \mathbf{E}[s_j]\mathbf{E}[s'_k]k'[i, j, k]$ ▷ Equation 3.37
14: $k \leftarrow k + \boldsymbol{\mu}[i]k_i$ ▷ Equation 3.31
15: **return** k

3.1.4 String kernels in NLP

The original benchmark application proposed in Lodhi et al. (2002) and Cancedda et al. (2003) is text classification on the Reuters-21578 corpus. The authors evaluate different string

kernels using SVMs as the learning algorithm. Lodhi et al. (2002) focused on applying a character level string kernels but found out that it was outperformed by simpler bag-of-words representations. The results showed in Cancedda et al. (2003) were more encouraging, with some combinations outperforming BOW models. Saunders et al. (2003) also experimented with the Reuters dataset but focused on theoretical connections between string kernels and Fisher kernels (Jaakkola et al., 2000), which are another class of kernels based on differences between probabilistic generative models.

Although the original formulations were motivated by text-based applications, string kernels have not been widely studied in NLP after these initial proposals. Some theoretical developments include the work of Rousu and Shawe-Taylor (2005), which proposes a sparse implementation based on hard matching in large alphabets, and Mahé and Cancedda (2008), which defines kernels between factored representations of symbols, such as words accompanied by their part-of-speech tags, for instance. From an application perspective, however, in NLP string kernels were not investigated beyond text classification applications.

It is worth noting that string kernels have been widely adopted in bioinformatics, more specifically in protein classification tasks. Examples of previous work in this include Leslie et al. (2003) and Igel et al. (2007). The last one is particularly interesting since it performs string kernel hyperparameter optimisation, which we will explore in Section 3.2.2. Their approach is unsupervised, relying on kernel alignment metrics while we focus on a supervised method using the response variables in a GP model.

3.2 Vectorised String Kernels

In this Section we develop a new algorithm for calculating a string kernel between two strings. The main idea is to redefine the internal calculations for k' in terms of vector and matrix multiplications. This new formulation, which we name *Vectorised String Kernels* (VSK), brings a series of benefits:

- Vector and matrix multiplications are massively parallelisable, meaning we can capitalise on recent advances in GPU processing for faster processing.
- It facilitates gradient implementations, which are necessary for hyperparameter optimisation under a GP model.
- It eases "batch mode" implementations, where we calculate kernel values for a whole set of string pairs.

- Finally, the new formulation not only admits the extensions showed in Sections 3.1.1 and 3.1.2 but also makes them more explicit.

The key idea behind the VSK is to unroll the recursion when calculate k_i' in Equation 3.34 and k_i'' in Equation 3.36. For k_i', we can do this by first defining the matrix $\mathbf{K}_i'$, which contain the values for k_i' for each symbol pair, and then encoding the recursion as a triangular matrix of gap decay values:

$$\mathbf{D}_s = \begin{bmatrix} 0 & \lambda_g^0 & \lambda_g^1 & \lambda_g^2 & \dots & \lambda_g^{|s|-2} \\ 0 & 0 & \lambda_g^0 & \lambda_g^1 & \dots & \lambda_g^{|s|-3} \\ 0 & 0 & 0 & \lambda_g^0 & \dots & \lambda_g^{|s|-4} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \lambda_g^0 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}. \tag{3.38}$$

Updates to $\mathbf{K}_i'$ for each subsequence length i can then be encoded by simply multiplying $\mathbf{D}_s$ with the corresponding matrix for k_i''. To obtain this matrix, which we call $\mathbf{K}_i''$, we use another triangular matrix of gap decays, $\mathbf{D}_{s'}$, containing the same structure as $\mathbf{D}_s$ but with shape $|s'| \times |s'|$. The final component is the similarity measure between symbols, which can be precalculated in a matrix $\mathbf{S}$. With these elements we can formally define the set of equations for the VSK as

$$\mathbf{S} = \mathbf{E}_s \mathbf{E}_{s'}^T, \tag{3.39}$$

$$\mathbf{K}_0' = \mathbf{1}, \tag{3.40}$$

$$\mathbf{K}_i' = \mathbf{D}_s \mathbf{K}_i'' \mathbf{D}_{s'}, \tag{3.41}$$

$$\mathbf{K}_i'' = \lambda_m^2 (\mathbf{S} \odot \mathbf{K}_{i-1}'), \tag{3.42}$$

$$k_i = \sum_{j,k} \lambda_m^2 (\mathbf{S} \odot \mathbf{K}_i')_{jk}, \tag{3.43}$$

$$\bar{k} = \boldsymbol{\mu}^T \mathbf{k}, \tag{3.44}$$

where $\mathbf{E}_s$ and $\mathbf{E}_{s'}$ are matrices of symbol embeddings for each string and $\odot$ is the Hadamard (element-wise) product.

The formulation above is equivalent to Equations 3.22 to 3.26, where k_i'' is not used. This can be seen by first noticing that $\mathbf{K}_i''$ is not needed since the recursion happens only on $\mathbf{K}_i'$. In other words, $\mathbf{K}_i' = \mathbf{D}_s \lambda_m^2 (\mathbf{S} \odot \mathbf{K}_{i-1}') \mathbf{D}_{s'}$, but we keep two separate terms for the sake of clarity. In Equation 3.24 the matches are discounted by $\lambda^{|s'|-j+2}$, which in the

vectorised form becomes individual elements of $\mathbf{D}_{s'}$ times λ_m^2. The $\mathbf{S}$ matrix corresponds to checking when symbol matches happen, possibly enabling soft matching if $\mathbf{S}$ is dense. Finally, $\mathbf{D}_s$ is needed for the recursion itself and corresponds to the $\lambda k_i'$ term in Equation 3.24. The equivalence of Equations 3.43 and 3.26 is straightforward: the elements from $\mathbf{K}_i'$ are weighted by the symbol matches and λ_m^2 and then collapsed into a single vector.

The VSK has complexity $O(n\ell^3)$, where $\ell = \max(|s|, |s'|)$, which is easy to see from the need of two matrix multiplications in Equation 3.41. It also arises from the fact we are not relying on k_i'' anymore, originally introduced to decrease complexity from cubic to quadratic, as explained in Section 3.1. However, in practice we still get performance gains due to vectorisation, as we will see in the next Section.

3.2.1 Runtime performance

To assess the impact of vectorisation in our proposed kernel we perform a set of benchmark experiments where we measure wall-clock time for a Gram matrix calculation using the original string kernel and the VSK. To simplify the comparison we use the following fixed settings:

- We employ characters as symbols with a one-hot encoding as the embedding;
- The alphabet is composed of all English ASCII letters, including uppercase (52 symbols in total);
- The maximum subsequence length is set to 5;
- 100 instances are generated randomly by uniformly sampling a character until reaching the desired string length.

We test our kernels with lengths ranging from 10 to 100. Experiments were ran in an Intel i7-4790 3.6GHz.

Figure 3.1 shows wall-clock time measurements as the string lengths increase. It is clear that the VSK is vastly faster than the original SK, with up to two orders of magnitude, even though both run on a single core. That said, we also believe these benchmarks should be interpreted carefully. While they definitely show the efficiency of VSK from an *empirical* perspective we can not make any performance claims from a *theoretical* one. Furthermore, while we made efforts to optimise the code, there is no guarantee either of our implementations is making the most of the underlying hardware. We leave a theoretically-grounded evaluation of these two kernels for future work.

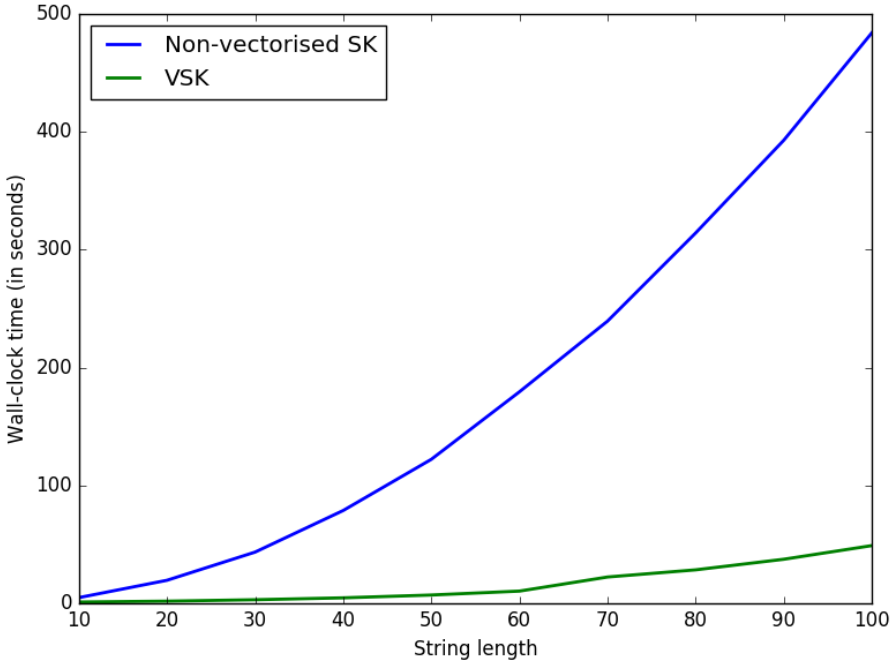


Fig. 3.1 Wall-clock time measurements for the standard SK and the VSK using different string lengths. Time is measured in seconds and correspond to the calculation of a 100 $\times$ 100 Gram matrix with random strings of a specific length.

Nevertheless, for the purposes of the applications pursued in this thesis, the results give enough evidence the VSK is a preferable alternative. This is particularly the case if we want to make use of multi-core architectures such as GPUs. We can easily extend the dimensionality of the data structures used in the VSK implementation and calculate the kernel between multiple string pairs at once, in a batch mode. This further enhances the performance of the kernel and makes better use of recent advances in multi-core hardware platforms.

3.2.2 Hyperparameter optimisation

In their general formulation, string kernels can be defined in a range of different hyperparameter settings. A kernel can be made more expressive by being parameterised in a more fine-grained way. Having different decay hyperparameters for sequence gaps and symbol matches is an example of such a procedure. This however, raises the question of how to perform efficient model selection for such kernels.

In Section 2.3.1 we saw how gradient-based methods can be employed for hyperparameter optimisation when the kernel is the covariance matrix of a GP. This procedure can also be used to optimise string kernels, an avenue that we pursue in this thesis. To perform

such a procedure, we need to derive the gradients of the VSK with respect to each of its hyperparameters.

From Equation 3.44 we can see that the gradient with respect to $\boldsymbol{\mu}$ is simply $\mathbf{k}$, i.e., the vector of kernel evaluations for each subsequence length. For λ_m we obtain the equations

$$\frac{\partial \mathbf{K}'_0}{\partial \lambda_m} = \mathbf{0}, \tag{3.45}$$

$$\frac{\partial \mathbf{K}'_i}{\partial \lambda_m} = \mathbf{D}_s \frac{\partial \mathbf{K}''_i}{\partial \lambda_m} \mathbf{D}_{s'}, \tag{3.46}$$

$$\frac{\partial \mathbf{K}''_i}{\partial \lambda_m} = 2\lambda_m (\mathbf{S} \odot \mathbf{K}'_{i-1}) + \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}_{i-1}}{\partial \lambda_m} \right), \tag{3.47}$$

$$\frac{\partial k_i}{\partial \lambda_m} = 2\lambda_m \sum_{j,k} (\mathbf{S} \odot \mathbf{K}'_i)_{jk} + \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}_i}{\partial \lambda_m} \right)_{jk}, \tag{3.48}$$

$$\frac{\partial \bar{k}}{\partial \lambda_m} = \boldsymbol{\mu}^T \frac{\partial \mathbf{k}}{\partial \lambda_m}. \tag{3.49}$$

And for λ_g we first define

$$\frac{\partial \mathbf{D}_s}{\partial \lambda_g} = \begin{bmatrix} 0 & 0 & \lambda_g^0 & 2\lambda_g^1 & 3\lambda_g^2 & \dots & (|s|-2)\lambda_g^{|s|-3} \\ 0 & 0 & 0 & \lambda_g^0 & 2\lambda_g^1 & \dots & (|s|-3)\lambda_g^{|s|-4} \\ 0 & 0 & 0 & 0 & \lambda_g^0 & \dots & (|s|-4)\lambda_g^{|s|-5} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & \lambda_g^0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{3.50}$$

and also $\dfrac{\partial \mathbf{D}_{s'}}{\partial \lambda_g}$ in a similar way. These matrices are then plugged into the equations

$$\frac{\partial \mathbf{K}'_0}{\partial \lambda_g} = \mathbf{0}, \tag{3.51}$$

$$\frac{\partial \mathbf{K}'_i}{\partial \lambda_g} = \frac{\partial \mathbf{D}_s}{\partial \lambda_g} \mathbf{K}''_i \mathbf{D}_{s'} + \mathbf{D}_s \frac{\partial \mathbf{K}''_i}{\partial \lambda_g} \mathbf{D}_{s'} + \mathbf{D}_s \mathbf{K}''_i \frac{\partial \mathbf{D}_{s'}}{\partial \lambda_g}, \tag{3.52}$$

$$\frac{\partial \mathbf{K}''_i}{\partial \lambda_g} = \lambda_m^2 \left(\mathbf{S} \odot \frac{\partial \mathbf{K}'_{i-1}}{\partial \lambda_g} \right), \tag{3.53}$$

$$\frac{\partial k_i}{\partial \lambda_g} = \lambda_m^2 \sum_{j,k} \left(\mathbf{S} \odot \frac{\partial \mathbf{K}'_i}{\partial \lambda_g} \right)_{jk}, \tag{3.54}$$

$$\frac{\partial \bar{k}}{\partial \lambda_g} = \boldsymbol{\mu}^T \frac{\partial \mathbf{k}}{\partial \lambda_g}. \tag{3.55}$$

The performance bottleneck for calculating the VSK gradients are the matrix multiplications at Equations 3.46 and 3.52, which means the worst case complexity is the same as the main VSK calculations, $O(n\ell^3)$, where $\ell = \max(|s|, |s'|)$. In practice, the gradients can be obtained at the same time the main kernel calculations happen, i.e., they can be part of the same algorithm. This further improves performance since many terms can be shared between the equations.

Having defined the gradients we can plug in the procedure of Section 2.3.1 and optimise the VSK hyperparameters. However, a natural question that arises in such a procedure is how much data is needed to learn these hyperparameters. To answer this question we run another set of benchmark experiments using synthetically generated data.

Our experiment protocol employs the following idea: if our data is sampled from a GP with a specific VSK then we should be able to use this data to train another GP with a VSK initialised at random and then recover the original hyperparameters by maximising the marginal likelihood is this sampled data.

More specifically, we define a GP with a VSK of subsequence order 3 with the following hyperparameter values:

$\lambda_g = 0.01$	$\lambda_m = 0.2$	$\sigma^2 = 0.01$
$\mu_1 = 1.0$	$\mu_2 = 0.5$	$\mu_3 = 0.25$

where σ^2 is the Gaussian likelihood noise. Then we randomly generated n strings of length 40 and calculate the Gram matrix of this GP (including σ^2, which is added to the diagonal elements). This Gram matrix is the covariance matrix of a multivariate Gaussian distribution with mean vector $\mathbf{0}$. By sampling from this Gaussian we can artificially create labels for the randomly generated strings and use this as labelled data.

The data sampled from the procedure above is used to train another GP with a VSK with *randomly initialised* hyperparameters. Considering our data is originally sampled from the model defined above, we should be able to retrieve the original hyperparameter values when maximising the marginal likelihood, *as long as there is enough data*. We run this procedure 20 times, using a fixed set of strings but sampling new labels every time.

The conditions above provide a controlled environment to check the modelling capacities of our approach since we know the exact distribution where the data comes from. The reasoning behind these experiments is that to be able to provide benefits in real tasks, where the data distribution is not known, our models have to be learnable in this controlled setting as well using a reasonable amount of data.

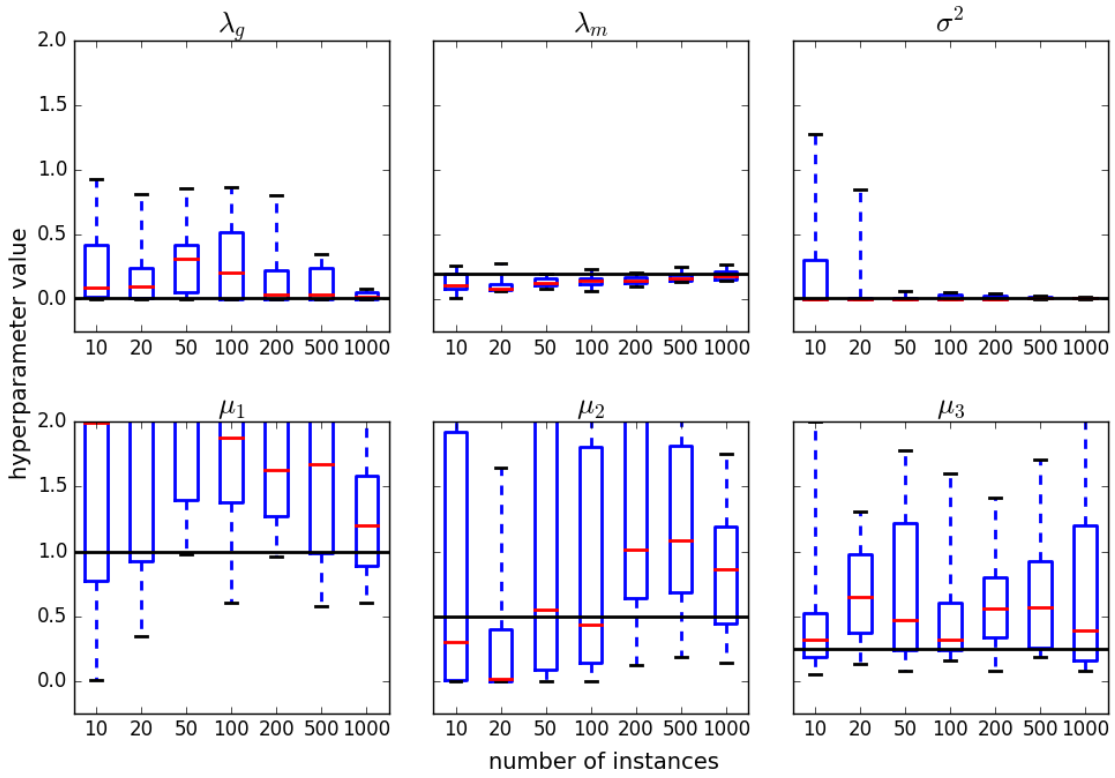


Fig. 3.2 String kernel hyperparameter optimisation results. For each hyperparameter its original value is shown as a black line in the plot. Each box corresponds to the obtained results using a specific dataset size, repeated 20 times. The red line shows the median value, the box limits correspond to the $[0.25, 0.75]$ quantiles and the whiskers show the maximum and minimum values obtained.

Figure 3.2 shows the results as dataset size increases. The likelihood Gaussian noise and the match decay λ_m are clearly recoverable even for small datasets. The gap decay λ_g is less stable but tends to be recoverable for datasets containing 200 instances and for 1000 instances it reaches its original value with high confidence.

The ngram coefficients $\boldsymbol{\mu}$ show a much more unstable behaviour compared to the other hyperparameters. A possible explanation for this is the presence of some level of over-specification. This means that very different coefficient values may reach similar marginal likelihoods, which in turn corresponds to multiple plausible explanations of the data. Overall, this behaviour shows that care should be taken when optimising these hyperparameters. Depending on the problem, this can mean imposing bounds during optimisation or even fixing them to sensible values, while freely optimising the more stable hyperparameters.

3.3 Tree Kernels

String kernels with soft matching via word embeddings can be effective in capturing semantic aspects of the data. Sometimes we are also interested in representing *syntactic* phenomena. For instance, in Quality Estimation we might want to check if the translated segment follows the structure of the target language, such as word order. String kernels in theory can capture some phenomena by allowing gappy n-grams as features but this can also result in them considering nonsensical sequences which would just add noise to the calculation.

If our text is composed of a single sentence we can overcome this issue by obtaining its *syntactic tree* through a parser. This way we can explicit restrict which syntactic structures should be taken into account when comparing two texts. Another advantage is that we can easily incorporate additional information from the treebanks where the parsers were trained on. This is very much like the idea of using pretrained word embeddings for soft matching.

Since now the sentences are represented as trees we have to employ appropriate kernels. Haussler's R-convolution framework can also be applied to trees, leading to the so-called tree kernels (Collins and Duffy, 2001). The idea is very similar to string kernels: they measure the similarity between two trees by counting how many tree *fragments* they share.

Different tree kernels can be obtained by employing different definitions for what a tree fragment is:

- If we consider only complete subtrees as fragments we obtain the Subtree Kernel (STK) (Vishwanathan and Smola, 2003). By complete we mean all fragment leaves correspond to leaves in the original tree.
- Relaxing this assumption but forcing fragments to contain complete grammatical rules generates *subset trees* and the corresponding kernel is the Subset Tree Kernel (SSTK) (Collins and Duffy, 2001).
- Finally, allowing fragments to have partial rules lead to Partial Tree Kernels (PTK) (Moschitti, 2006a). This means that when checking if two rules match some child symbols may be omitted.

Figure 3.3 shows an example of a constituency tree and the corresponding fragment space according to the kernels we introduced. In this thesis we focus on the SSTK since we employ constituency trees as representations and previous work argued the SSTK is more suitable for these kind of trees (Hardmeier, 2011). Nevertheless, some of the proposed ideas could also be applied in other tree kernels.

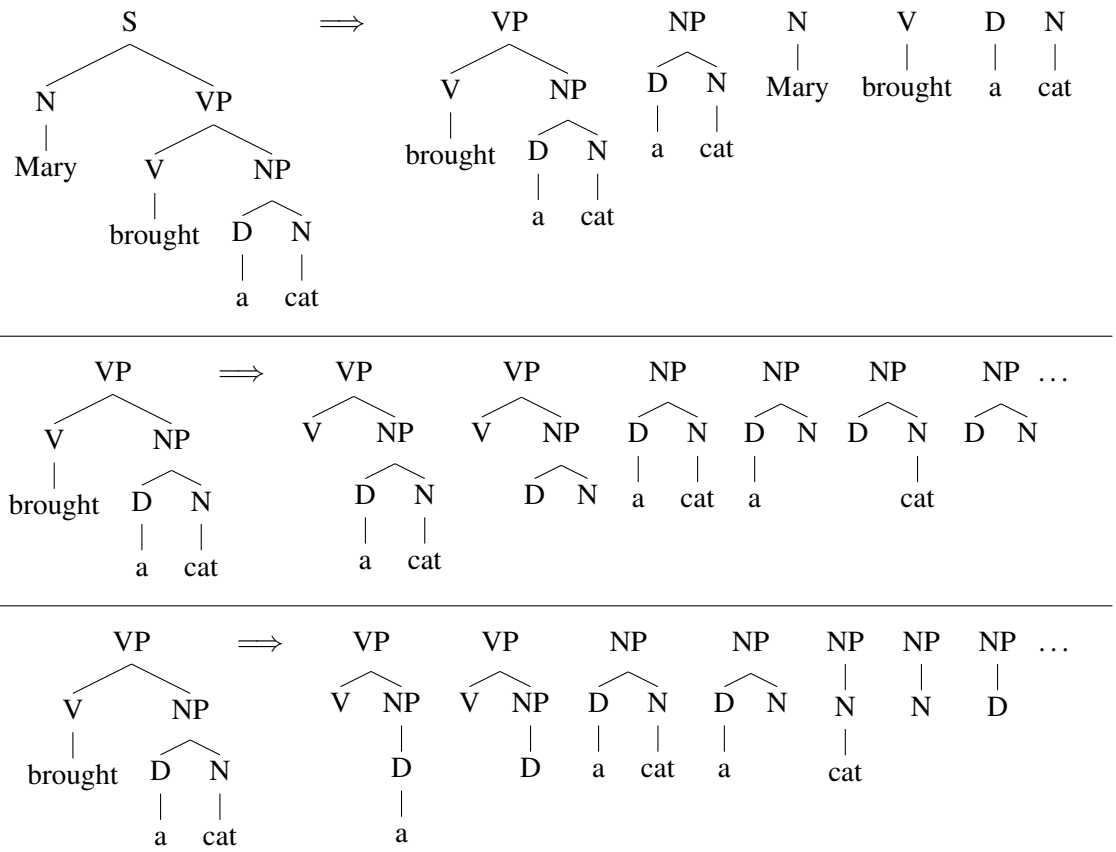


Fig. 3.3 Examples of tree fragments. On the top we show all fragments which compose the underlying feature space for the Subtree Kernel. The middle row shows some fragments from a Subset Tree Kernel. On the bottom we show fragments extracted from a Partial Tree Kernel. Figure adapted from Moschitti (2006b).

Similar to string kernels, it is possible to devise efficient dynamic programming algorithms to calculate the SSTK. Here we detail this DP formulation, following Collins and Duffy (2002). Let $\mathcal{F}$ be the set of all tree fragments and N and N' be the sets of nodes in t and t', respectively. Define the indicator function $I_f(n)$ to be 1 if a fragment f has n as its root node and 0 otherwise. The SSTK is defined as

$$k(t,t') = \sum_{f\in\mathcal{F}}\sum_{n\in N}\sum_{n'\in N'} I_f(n)I_f(n') \tag{3.56}$$

$$= \sum_{n\in N}\sum_{n'\in N'}\sum_{f\in\mathcal{F}} I_f(n)I_f(n') \tag{3.57}$$

$$= \sum_{n\in N}\sum_{n'\in N'} \Delta(n,n'). \tag{3.58}$$

Intuitively, for each pair of nodes between t and t', Δ counts the number of common fragments between them. Notice that two nodes can be the root of multiple fragment matches, for instance if $t = t'$. This is why the summation over the fragment space is required.

Naively calculating Δ is impractical due to the exponential size of $\mathcal{F}$: this is where DP is employed. Let $\mathrm{rule}(n)$ be the grammar rule at node n and $\mathrm{preterm}(n)$ be a binary variable which is true if n is a pre-terminal node (contains only a single terminal leaf). With this, we define Δ recursively as

$$\Delta(n, n') = \begin{cases} 0 & \mathrm{rule}(n) \neq \mathrm{rule}(n'), \\ 1 & \mathrm{rule}(n) = \mathrm{rule}(n') \wedge \mathrm{preterm}(n), \\ g(n, n') & \text{otherwise}, \end{cases} \tag{3.59}$$

$$g(n, n') = \prod_{i=1}^{|n|} (1 + \Delta(c_n^i, c_{n'}^i)), \tag{3.60}$$

where $|n|$ is the number of children of node n and c_n^i is the i^{th} child node of n. The first two cases are trivially correct. For the third case, remember that Δ counts fragment matches between the nodes. A common fragment between n and n' can be formed by taking the rule at node n or n', together with a choice at each child to attach either just the non-terminal node corresponding to the child or any common fragments at that child. Therefore, for each child i there are $(1 + \Delta(c_n^i, c_{n'}^i))$ choices.

This formulation was extended by Moschitti (2006b), resulting in the final SSTK form we employ:

$$\Delta(n, n') = \begin{cases} 0 & \mathrm{rule}(n) \neq \mathrm{rule}(n'), \\ \lambda & \mathrm{rule}(n) = \mathrm{rule}(n') \wedge \mathrm{preterm}(n), \\ \lambda g(n, n') & \text{otherwise}, \end{cases} \tag{3.61}$$

$$g(n, n') = \prod_{i=1}^{|n|} (\alpha + \Delta(c_n^i, c_{n'}^i)), \tag{3.62}$$

where $\lambda \in (0, 1]$ is a decay hyperparameter that penalises large fragment contributions, similar to what is done in string kernels, and $\alpha \in [0, 1]$ is an hyperparameter that can select between considering subtrees or subset trees as fragments. Originally, Moschitti (2006b) uses either 0 or 1 as values for α, which corresponds to a hard selection between the Subtree

Kernel and the SSTK. Here we define it to be a continuous value to be optimised, letting the kernel itself decide the fragment representation that better fits the data.

The SSTK complexity in the worst case is $O(NN')$ (Collins and Duffy, 2001). However, assuming a hard match approach between tree nodes, the average runtime in practice is linear in either N or N', whichever is larger. This is because most grammar rules will not match in natural language constituency trees. Considering this behaviour, Moschitti (2006b) developed a different algorithm based on a heuristic that relies on representing trees as ordered lists of rules, which can be computed offline, before any kernel computation. The SSTK is then defined over identical rule pairs, which are extracted using a binary search procedure. This does not alter the dynamic programming formulation as it only skips calculations when $\Delta = 0$. As showed in Moschitti (2006b), this alternative algorithm can bring vast performance gains in practice.

3.3.1 Symbol-aware Subset Tree Kernel

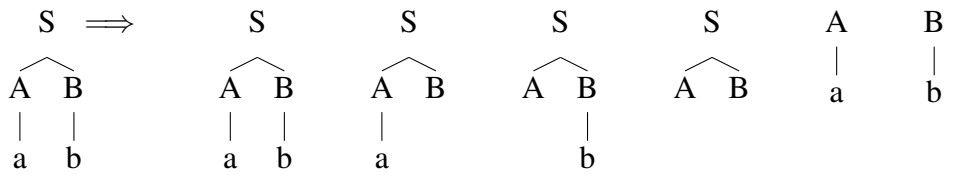
In Section 3.1.1 we showed how it is possible to split decay hyperparameters in string kernels to achieve larger model capacity. A similar idea can be applied in the SSTK. Here we propose to have *symbol-specific* decays, i.e., one λ for each non-terminal symbol in the grammar. The same extension can also be applied to the α hyperparameter. This idea effectively gives different weights to tree fragments with different root labels. For instance, if a task benefits more from signals coming from noun phrases this can be encoded by assigning higher values for their specific hyperparameters λ_{NP} and α_{NP}. We name this new formulation *Symbol-aware Subset Tree Kernel* (henceforth, SASSTK). Its calculation uses a similar recursive formula to the SSTK:

$$\Delta(n, n') = \begin{cases} 0 & \text{rule}(n) \neq \text{rule}(n') \\ \lambda_x & \text{rule}(n) = \text{rule}(n') \wedge \text{preterm}(n) \\ \lambda_x g_x(n, n') & \text{otherwise,} \end{cases} \tag{3.63}$$

$$g_x(n, n') = \prod_{i=1}^{|n|} (\alpha_x + \Delta(c_n^i, c_{n'}^i)), \tag{3.64}$$

where x is the symbol at node n.

The SASSTK can be interpreted as a generalization of the SSTK: we can recover the latter by tying all λ and setting all $\alpha = 1$. By employing different hyperparameter values for each specific symbol, we can effectively modify the weights of all fragments where the



	$\lambda_S = 1$ $\lambda_A = 1$ $\lambda_B = 1$	$\lambda_S = 0.5$ $\lambda_A = 1$ $\lambda_B = 1$	$\lambda_S = 2$ $\lambda_A = 1$ $\lambda_B = 1$
A → a	1	1	1
B → b	1	1	1
S → A B	1	0.5	2
S → (A a) B	1	0.5	2
S → A (B b)	1	0.5	2
S → (A a) (B b)	1	0.5	2
$k(t,t)$	6	3	18

Fig. 3.4 Resulting fragment weighted counts for an example SASSTK calculation. On the top we show an instance of a tree t, with its corresponding fragments. The table on the bottom shows the results of $k(t,t)$ for different set of hyperparameters with the explicit weighted counts.

symbol appears as root node. Figure 3.4 shows an example where we unrolled a kernel computation into its corresponding feature space, showing the resulting weighted counts for each feature.

The SASSTK hyperparameters can in theory be set by hand if we have good prior knowledge about which non-terminal symbols contain more signal. However, it is much more interesting to learn these hyperparameters from data, following the ideas explained in Section 2.3.1. In fact, the main point of enhancing the original SSTK using more complex parameterisations is to employ GP-based model selection procedures, which we will perform in the next Section.

3.3.2 Hyperparameter optimisation

Similar to what was proposed for string kernels in Section 3.2.2, our main goal is to be able to optimise tree kernel hyperparameters inside a GP model via gradient-based methods. Here we extend the tree kernel formulation to allow the computation of gradients with respect to the hyperparameters.

From Equation 3.58 we know that the kernel is a double summation over the Δ function. Therefore all gradients are also double summations, but over the gradients of Δ. We can obtain these in a vectorised way, by considering the gradients of the hyperparameter vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ over Δ. Let k be the number of symbols considered in the model and $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$ be k-dimensional vectors containing the respective hyperparameters. We start with the $\boldsymbol{\lambda}$ gradient:

$$\frac{\partial \Delta(n, n')}{\partial \boldsymbol{\lambda}} = \begin{cases} 0 & \text{rule}(n) \neq \text{rule}(n'), \\ \boldsymbol{u} & \text{rule}(n) = \text{rule}(n') \wedge \text{preterm}(n), \\ \dfrac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\lambda}} & \text{otherwise}, \end{cases} \tag{3.65}$$

where x is the symbol at n, g_x is defined in Equation 3.64 and $\boldsymbol{u}$ is the k-dimensional unit vector with the element corresponding to symbol x equal to 1 and all others equal to 0. The gradient in the third case is defined recursively,

$$\frac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\lambda}} = \boldsymbol{u} g_x + \lambda_x \frac{\partial g_x}{\partial \boldsymbol{\lambda}} \tag{3.66}$$

$$= \boldsymbol{u} g_x + \lambda_x \sum_{i=1}^{|n|} \frac{g_x}{\alpha_x + \Delta(c_n^i, c_{n'}^i)} \frac{\partial \Delta(c_n^i, c_{n'}^i)}{\partial \boldsymbol{\lambda}}. \tag{3.67}$$

The $\boldsymbol{\alpha}$ gradient is derived in a similar way,

$$\frac{\partial \Delta}{\partial \boldsymbol{\alpha}} = \begin{cases} 0 & \text{rule}(n) \neq \text{rule}(n') \vee \text{preterm}(n) \\ \dfrac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\alpha}} & \text{otherwise}, \end{cases} \tag{3.68}$$

and the gradient at the second case is also defined recursively,

$$\frac{\partial(\lambda_x g_x)}{\partial \boldsymbol{\alpha}} = \lambda_x \frac{\partial g_x}{\partial \boldsymbol{\alpha}} \tag{3.69}$$

$$= \lambda_x \sum_{i=1}^{|n|} \frac{g_x}{\alpha_x + \Delta(c_n^i, c_{n'}^i)} \left(\boldsymbol{u} + \frac{\partial \Delta(c_n^i, c_{n'}^i)}{\partial \boldsymbol{\alpha}} \right). \tag{3.70}$$

The worst case complexity for gradient calculations is the same as the main kernel algorithm, $O(NN')$, and it can also benefit from the speedup heuristic proposed by Moschitti (2006b). Similar to what we do with the VSK in Section 3.2.2, the SASSTK gradients can be calculated at the same time as Δ to improve performance since they all share many terms in their derivations. Finally, we can easily obtain the gradients for the original SSTK by letting $\boldsymbol{u} = 1$.

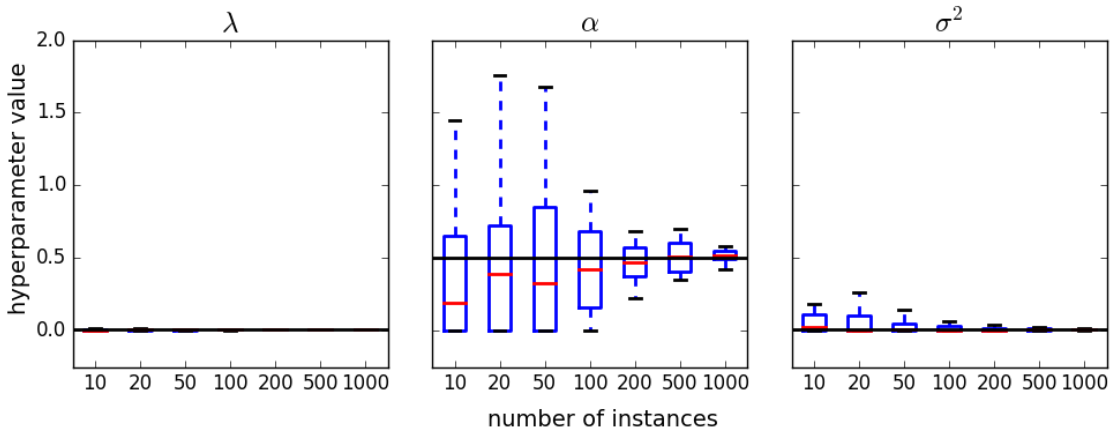


Fig. 3.5 SSTK hyperparameter optimisation results. For each hyperparameter its original value is shown as a black line in the plot. Each box corresponds to the obtained results using a specific dataset size, repeated 20 times. The red line shows the median value, the box limits correspond to the $[0.25, 0.75]$ quantiles and the whiskers show the maximum and minimum values obtained.

As in the VSK, the question of how much data is needed to learn hyperparameters also arises in the (SA)SSTK. Therefore we proceed to perform similar benchmarks to address this question. However, instead of generating random structures, we use a subset of annotated natural language sentences from the Penn Treebank (Marcus et al., 1993), obtained from the NLTK toolkit (Bird et al., 2009).

The protocol follows a similar setting to the one described in Section 3.2.2, where we sample labels from a GP with a tree kernel with fixed hyperparameter values and use these to train a different GP with a different tree kernel with randomised hyperparameter values. We first start by benchmarking the SSTK, using $\lambda = 0.01$, $\alpha = 0.5$ and $\sigma^2 = 0.01$ as fixed values for the GP which originate the labels. As in the VSK experiments, we resample labels and optimise the new GP 20 times. We repeat this procedure for increasing subsets of trees.

Figure 3.5 shows the results of our benchmarks. The noise and decay hyperparameters are easily recovered even for small dataset sizes. The branch hyperparameter needs more data but the optimisation procedure is able to find the original values with high confidence for dataset with 200 instances. These results are encouraging since they are much more stable than the ones obtained with the VSK and also because they employ real world natural language trees as inputs (only the response variables are synthetic).

Next we benchmark the SASSTK. The experimental protocol is similar to the one used in the SSTK benchmarks, except that we now use a kernel with specific hyperparameters for noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). The choice of these

symbols is arbitrary: the idea is to define a distribution over the response variables which can not be encoded by a simple SSTK. As for the hyperparameter values we fix $\sigma^2 = 0.01$ and use the following for the tree kernel ones:

$\lambda_{def} = 0.01$	$\lambda_{NP} = 0.2$	$\lambda_{VP} = 0.1$	$\lambda_{PP} = 0.5$
$\alpha_{def} = 0.5$	$\alpha_{NP} = 0.6$	$\alpha_{VP} = 0.8$	$\alpha_{PP} = 0.9$

where λ_{def} and α_{def} correspond to tied default hyperparameters for all other grammar symbols.

The results are shown on Figure 3.6. The noise and default decay hyperparameters are again easily recoverable even for small datasets. Symbol specific decay hyperparameters can be retrieved with high confidence for datasets with 500 instances, with the exception of λ_{PP}, which exhibits larger variance. The α hyperparameters tend to be more easily recovered as the number of instances increases but they still exhibit some degree of uncertainty even when using 1000 instances.

The SASSTK is flexible enough to accommodate a large number of symbol-specific hyperparameters. However, these benchmarks show that allowing rich parameterisations might not be justified considering we are dealing with datasets where annotations are scarce. Therefore we believe tying most of the hyperparameters is a sensible choice in the tasks we pursue in this thesis. Nevertheless, our SASSTK can still benefit from untying hyperparameters for some symbols. Deciding which ones should be untied is an open question but we foresaw some possibilities:

Using linguistic prior information: if we know beforehand that some syntactic structures such as noun phrases will carry more signal to our task we can use this knowledge to justify the untying for the corresponding symbols;

Using frequency information: we can inspect the data and untie hyperparameters for symbols that appear more frequently, up to a threshold.

3.3.3 Normalisation

Values obtained from a tree kernel can be very dependent on the size of the trees. This is usually not desirable as it may bias the underlying learning algorithm, especially if the training data has a lot of variation on the tree sizes (Collins and Duffy, 2001). This can be

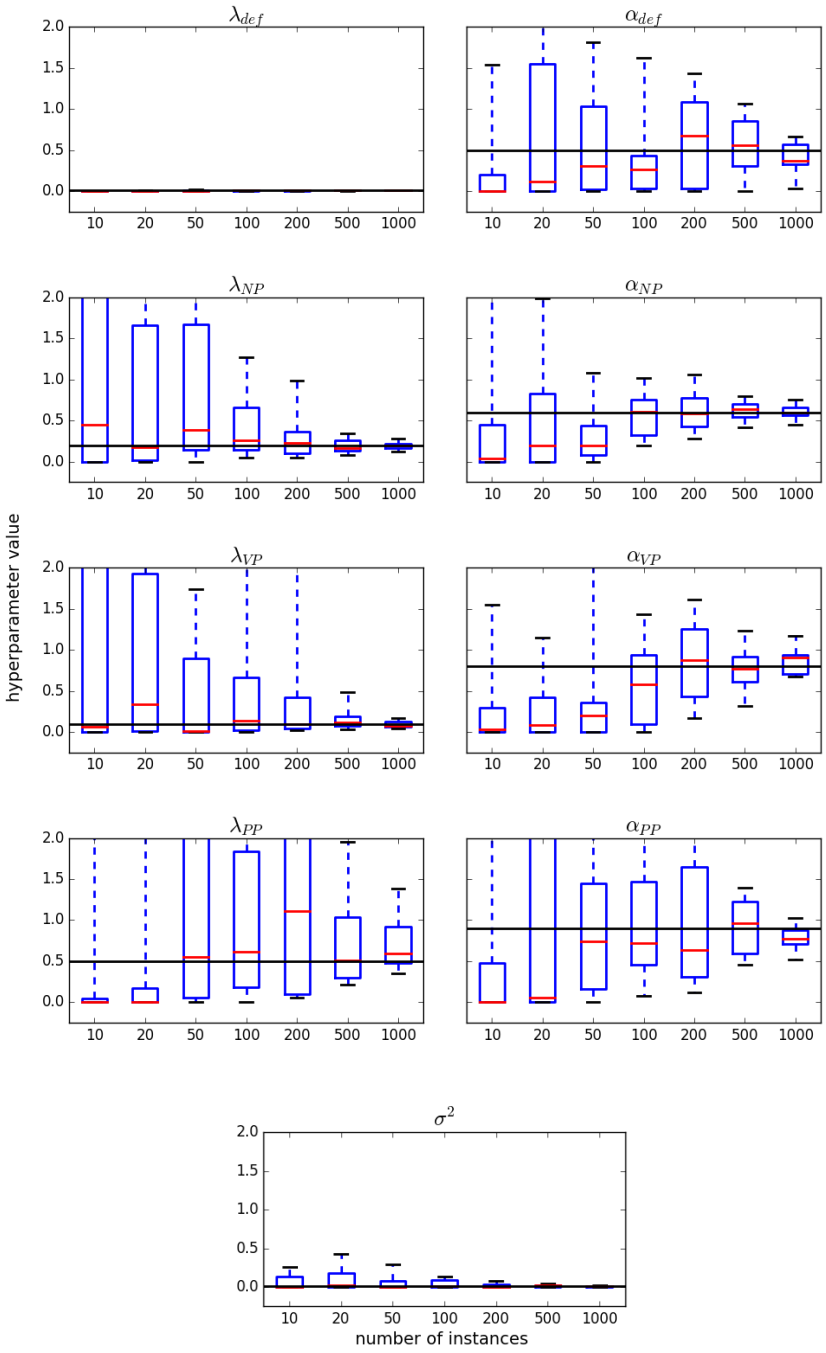


Fig. 3.6 SASSTK hyperparameter optimisation results. For each hyperparameter its original value is shown as a black line in the plot. Each box corresponds to the obtained results using a specific dataset size, repeated 20 times. The red line shows the median value, the box limits correspond to the $[0.25, 0.75]$ quantiles and the whiskers show the maximum and minimum values obtained.

mitigated by employing the normalisation property we showed in Section 2.3.5:

$$\hat{k}(t, t') = \frac{k(t, t')}{\sqrt{k(t, t)k(t', t')}} \tag{3.71}$$

where k corresponds to the underlying tree kernel. Normalisation results in a score between 0 and 1, independent of the size of t or t'.

To apply normalised tree kernels within the GP model selection procedure we need to introduce its gradients,

$$\frac{\partial \hat{k}(t, t')}{\partial \boldsymbol{\theta}} = \frac{\frac{\partial k(t, t')}{\partial \boldsymbol{\theta}}}{\sqrt{k(t, t)k(t', t')}} - \hat{k}(t, t')\frac{\frac{\partial k(t, t)}{\partial \boldsymbol{\theta}}k(t', t') + k(t, t)\frac{\partial k(t', t')}{\partial \boldsymbol{\theta}}}{2k(t, t)k(t', t')}, \tag{3.72}$$

where $\boldsymbol{\theta}$ corresponds to the set of kernel hyperparameters. In theory, normalisation requires extra computations since it uses $k(t, t)$ and $k(t', t')$, as well as its gradients. In practice, model selection requires a full Gram matrix, which means these values can be retrieved from the Gram matrix diagonal, resulting in negligible speed overhead at training time. Finally, normalisation can be applied to any kernel, including the string kernel variations we introduced. This can be useful if the training data shows variation in text length.

3.3.4 Tree kernels in NLP

Unlike string kernels, tree kernel have been heavily used in NLP in the past. The original paper by Collins and Duffy (2001) that proposed the SSTK employed it in a parser reranking framework. They train a syntactic parser using the ATIS subset of the Penn Treebank and use a voted perceptron to rerank the top 100 trees obtained from a parser. The perceptron uses a SSTK as the underlying feature representation and is able to give better syntactic trees compared to the single 1-best output from the parser. The same idea was applied in a later paper on the full Penn Treebank and, at the time, a state-of-the-art parser (Collins and Duffy, 2002). Their findings were similar as the previous paper, with the voted perceptron reranking giving better results than the 1-best parser output.

Our implementation is based on the formulation proposed by Moschitti (2006b), which uses tree kernels within an SVM to predict semantic roles in sentences. His results using tree kernels only were comparable but still below a baseline with hand-crafted features. However, he got the best results by combining both representations using a combination of kernels, which shows they are complementary to some degree. This also shows how one can easily capitalise on state-of-the-art features in a kernel framework. As for specific tree kernel

versions, he experimented with both Subtree Kernel and the SSTK, with mixed results. This furthers motivate our approach, which is able to learn a tree kernel that is effectively between both representations.

Other applications where tree kernels were used include Question Classification (Croce et al., 2011; Moschitti, 2006a), Textual Similarity measuring (Severyn et al., 2013) and Relation Extraction (Bloehdorn and Moschitti, 2007; Plank and Moschitti, 2013). In regression, Hardmeier (2011) and Hardmeier et al. (2012) use tree kernels for Quality Estimation. Their findings are similar to Moschitti (2006b): the use of standalone tree kernels gives comparable but inferior performance to baselines while combinations which incorporate hand-crafted features give the best results.

3.4 Discussion

In this Chapter we introduced kernels for strings and trees, two common representations found in natural language applications. By making use of the kernel trick and dynamic programming procedures, these kernels can capture structural information which would not be computationally feasible through explicit feature enumeration. Their flexibility allows one to extend their capacity by defining more fine-grained parameterisations but raises the issue of how to optimise them in a efficient way.

Here we propose to capitalise on GP-based model selection procedure through empirical Bayes to optimise the hyperparameters of structural kernels. For strings we develop a new vectorised version of string kernels, namely VSK, and derive its gradients. This new algorithm also facilitates implementation in multicore architectures. We also assess the learnability of such kernels by performing an empirical study on synthetic data, showing that we are able to learn most hyperparameters even in the scarce data scenarios we are interested in. A remaining question is why the VSK shows large speedup gains in practice even though it shows higher complexity compared to the non-vectorised version.

To deal with tree representations, we start from a previously proposed tree kernel (SSTK) and generalise it to enable symbol specific hyperparameters, resulting in the SASSTK. We derive its gradients to enable GP-based optimisation and perform another empirical evaluation of its learnability. Our findings show that it can effectively learn the hyperparameters in scarce data settings but care should be taken when extending the model capacity since the higher number of hyperparameters makes the kernel more data-hungry.

While empirical experiments with synthetic data can give insights on the capacity of these kernels and their data requirements while also providing grounds for their usability it

still uses artificially generated response variables. In the following Chapters, we propose to use these kernels alongside the methods showed in Chapter 2 in three real world NLP applications. This combination "closes the loop" with respect to the thesis aims explained in Section 1.1: while structural kernels provide us with rich text representations, the GP framework allows us to have a well calibrated model of uncertainty in the response variables.

Chapter 4

Emotion Analysis

Being a fundamental part of human nature, emotions are a vastly studied subject in psychology and behavioural sciences. According to Oatley (1989), they arise from the need to adapt to unpredictable situations, where rational behaviour is less useful. Therefore, he argues that emotions should always be connected to some kind of action or outward realisation, which can appear in the form of facial expressions (Ekman, 1993; Keltner and Ekman, 2003) or vocal intonations (Murray and Arnott, 1993), for instance.

Given that emotions can be connected to human actions or cues, it is natural to wonder if they can be realised in the form of human language. Psycholinguistic studies say that this is indeed the case and a range of works aimed at investigating the connections between words and emotional states (Clore et al., 1987; Ortony et al., 1987). These studies have led to the definition of emotion taxonomies, which are sets of words that can be classified according to specific emotional attributes.

The connections between human emotion and natural language can be realised in two ways. While we use natural language to express our feelings, reading a text fragment or listening to an utterance can also affect how we feel. The latter has motivated emotion analysis from an applied, NLP perspective, where the goal is to build models that infer emotions from text data. Strapparava and Mihalcea (2007) cite a number of applications where such models could be useful:

Opinion Mining: categorising texts according to their affective load can provide a more fine-grained output to opinion mining systems, which in turn can be useful for market analysis, for instance.

	anger	disgust	fear	joy	sadness	surprise
Storms kill, knock out power, cancel flights	3	9	82	0	60	0
Morrissey may cheer up Eurovision	0	0	2	61	0	10
Archaeologists find signs of early chimps' tool use	0	0	2	23	0	64
Republicans plan to block Iraq debate	60	17	0	0	37	7
European Space Agency	0	0	0	2	0	0

Table 4.1 Emotion annotation examples, taken from the English news headlines dataset.

Computer Assisted Creativity: this ranges from personalised product adverts to a more ambitious goal of having a computational model of creativity that generates affective text.

Human-Computer Interaction: a model of emotions in text could help agents and chatbots to generate more natural conversations, for instance.

In this Chapter, we focus on the problem of inferring emotions from natural language text. The main approach used in this task is through supervised learning using human annotated text fragments. Table 4.1 shows some examples from the dataset we use in our experiments. The texts are news headlines and the scores were obtained from explicit annotations made by human judges, in the $[0, 100]$ range. More details about the dataset are given in Section 4.2.

Given that labels are obtained through human annotations they can exhibit substantial amount of bias and noise. These can be attributed to a series of factors, from the judge's particular emotional experiences to the annotation guidelines and interfaces. When modelling the relations between text and emotional scores we would like to take into account the inherent uncertainty that arises from these factors. This makes Gaussian Processes a natural fit for modelling this kind of data. Here we explore the combination of GP models with the string kernels introduced in Section 3.2 to build models that can not only predict emotions for unseen text but also provide insights in the relation between individual emotions.

Section 4.1 gives an overview of related work in this field, focusing on NLP contributions but connecting with other areas as well. We give a detailed description of the dataset we use for this task in Section 4.2, including preprocessing steps. On Section 4.3 we give our first contribution, where we evaluate the combination of GPs and string kernels to build models for each emotion individually. Section 4.4 introduces our second contribution: a joint

emotion model using multi-task GPs. The model can provide better use of the available data by incorporating all emotion labels as a vector of response variables and also can highlight connections between individual emotions through the inspection of specific hyperparameters. We wrap up with a discussion in Section 4.5.

4.1 Related Work

The first studies that addressed Emotion Analysis from an NLP perspective focused at the word level. These were motivated by the work of Ortony et al. (1987) and Clore et al. (1987), which aimed at building emotion taxonomies from what they call *affective lexica*. The authors argue that previous work that intended to study connections between words and emotional states actually use the concept of *affect* instead, which is broader. An example is the word "abandoned", which carries affectional load but does not necessarily refer to any emotion, unlike the word "angry" (Ortony et al., 1987). Therefore, they propose to focus on words that have direct connections to emotional states and proceed to build and evaluate emotion taxonomies.

The psycholinguistic work of Ortony and Clore was used as a background theory to enhance WordNet (Miller, 1995), an English lexical database which group content words (nouns, verbs, adjectives and adverbs) into clusters according to their sense. These clusters are called *synsets* and represent a single concept that can be realised into one or more words. Additionally, synsets are linked through conceptual-semantic relations, resulting in a large graph with synsets as vertices and relations as edges. Strapparava and Valitutti (2004) created WordNet-Affect, which extends WordNet by assigning the emotional labels proposed by Ortony et al. (1987) to synsets. This extension was built in a semi-automatic way, starting from a set of core synsets which were manually annotated and propagating these labels automatically using the conceptual relations imbued within the database. WordNet-Affect was later extended by Strapparava et al. (2006) by assigning vector representations to synset words and emotional concepts, allowing one to obtain weighted emotional loads for unseen words.

Subsequent work in this field proceeded to address how emotions are connected to larger text fragments beyond single lexical items. Alm et al. (2005) studied emotions in the context of children's fairy tales and developed a corpus annotated at the sentence level. Unlike WordNet-Affect, the authors used a different taxonomy proposed by Ekman (1993), which is composed of six emotions: *anger, disgust, fear, joy, sadness* and *surprise*. The annotations were obtained using 2 and 3-class granularities, varying from negative to positive in each of

the emotions. The authors then proceed to develop machine learning methods based on a set of hand-crafted features, reporting encouraging results.

Strapparava and Mihalcea (2007) proposed a shared task at the SemEval conference, called Affective Text. They introduced a corpus of English news headlines with fine-grained annotations in the $[0, 100]$ range, following the Ekman taxonomy of six emotions. Multiple annotations for each sentence were obtained from human judges and the final labels for each emotion were averaged. The example instances shown on Table 4.1 were taken from this dataset. Additionally, the corpus also provide a general negative/positive valence label, in the $[-100, 100]$ range. The focus of the shared task was on unsupervised models so the labels were not made available until after the task ended. Submitted systems used a range of information sources, including lexical resources such as WordNet-Affect but also implicit ones obtained from web search engine queries. The ideas obtained from the shared task submissions were used as a basis for subsequent work (Strapparava and Mihalcea, 2008), which used a series of corpus-based approaches to infer emotion labels from the same dataset.

Recent work focused on analysing emotions in music. This subject is widely studied and usually combine information from song lyrics with acoustic signals from song melodies (Kim et al., 2010; Yang and Lee, 2009). However, these previous works focused on obtaining emotional labels at the song level, usually in the form of binary indicators. Mihalcea and Strapparava (2012) were the first ones that analysed emotions from song lyrics at the line level, providing a corpus similar to the one used in the Affective Text shared task. A total of 100 songs and 4,976 lines are present in the corpus, each line being annotated by the six Ekman emotions using a $[0, 10]$ numerical scale. Following previous work, the authors develop a model which uses a combination of textual and musical features, which outperforms models trained either on textual or musical features only.

While the Ekman taxonomy is the most common in NLP work, other taxonomies have been also investigated. Mohammad and Turney (2011) propose the creation of emotion lexica through crowdsourcing, called EmoLex. Their main motivation is that other resources such as WordNet-Affect are very limited, in the order of hundreds of words, while this works aims at creating a lexicon in the order of tens of thousands of words. For emotion annotation they use a taxonomy proposed by Plutchik (1980), which is composed of the six Ekman emotions plus *anticipation* and *trust*. This taxonomy also argues that emotions can be organised into opposing pairs, such as *joy/sadness* and *anger/fear*.

Emotion Analysis bears some similarities to Sentiment Analysis, which, unlike the former, is a much more active area in NLP. Pang and Lee (2008) provide a survey that defines most of the concepts used in the field, while examples of recent work include Wang and Manning

(2012) and Socher et al. (2013). The main difference between the tasks is that Sentiment Analysis concerns about valence prediction only (positive/negative labels). It is also usually focused on coarse-grained labels and text classification approaches, unlike the fine-grained emotion labels that are provided in some of the datasets described above.

Finally, Emotion Analysis also bears connections to the field of Affective Computing (Picard, 1997), which advocates that computational agents must have affective traits in order to show intelligence and be effective in real world environments. This is very much in line with the observations made by Oatley (1989) with respect to the connection between emotions and actions. In this sense, Emotion Analysis can provide the means to equip computational agents with affection by the use of natural language, either in recognition or in generation.

4.2 Datasets and Preprocessing

For our experiments we use the dataset provided by the Affective Text shared task in SemEval 2007 (Strapparava and Mihalcea, 2007). It is composed of a set of news headlines taken from major newspapers such as New York Times, CNN and BBC News and also from Google News search engine. The main motivation for focusing on such a domain is because these sentences tend to have stronger emotional content since they describe major events and also are written in a style aiming at attracting the attention of the readers.

Annotation was performed through manual labelling and correspond to emotional strength values for each of the six Ekman emotions, in $[0, 100]$ interval, where 0 means total lack of the corresponding emotion and 100 corresponds to maximum emotional load. For the shared task, the dataset is divided into a "trial set" and a "test set". Here we join both sets into a single one, totalling 1,250 instances.

Our experiments involve the use of word embeddings to represent each word in a sentence. For this, we first preprocessed each sentence using the NLTK Penn Treebank tokeniser and downcased each word. Then, during model training, each word is represented by a 100-dimensional GloVe embedding (Pennington et al., 2014). We use word vectors trained on a combination of Wikipedia and Gigaword corpora, containing six billion tokens and a vocabulary size of 400,000 words.[1]

[1] Available at http://nlp.stanford.edu/projects/glove

4.3 String Kernels for Emotion Analysis

To build a model for Emotion Analysis we need to define a representation for the textual inputs. In Chapter 1 we mentioned two approaches for that: bag-of-words (BOW) and averaged embeddings. Using BOW is straightforward and can be effective in large datasets. However, representing words as one-hot independent vectors would require that all signal comes from the emotion scores, which is problematic considering we are dealing with scarce data.

Word embeddings are a simple way to provide external knowledge from large corpora into our task. By adopting a low-dimensional word representation they can encode similarities between words. This can be potentially helpful for our task since emotion scores related to a word or set of words in the training data can propagate to unseen words at test time, since they can be related through similar (or dissimilar) embeddings. A simple way to extrapolate this to the sentence level is by averaging the embeddings of all words present in the sentence and we use this approach as our main baseline.

The main drawback of the averaging method is that, much like BOW, it ignores word order. This can be too limiting if signal is present in larger, potentially non-contiguous word sequences. The string kernels introduced in Section 3.2 have the ability to encode such sequences implicitly and can also incorporate word embeddings through soft matching. In this Section we propose to use string kernels in conjunction with GPs for Emotion Analysis.

4.3.1 Experimental settings

Our main model is a GP with a VSK of order 5, i.e., a vectorised string kernel considering subsequences composed of up to 5 lexical items. We compare this approach with a series of baseline GP models trained on averaged word embeddings

Linear kernel: this is the baseline that is most directly related to our proposed approach since the string kernel is also composed of linear operations.

Non-linear kernels: we also compare with GPs trained on a set of non-linear kernels, including SE, Matèrn $3/2$ and Matèrn $5/2$. These are less comparable to our approach but helps to put it into a broader perspective.

All GP models are optimised using empirical Bayes. While it is possible to employ ARD for these kernels, in our experiments it showed worse performance compared to their corresponding isotropic version. This is likely to be due to overfitting since with ARD our models

end up having a 100-dimensional lengthscale vector to optimise. Therefore, we restrict our analysis to isotropic kernels. The GP models also have an added bias kernel to account for the scaling of the response variables and the kernels used in the averaging models have an additional variance hyperparameter, as explained in Section 2.3.5.

We also provide a few non-Bayesian baselines:

Ridge regression: a linear regression model with squared norm regularisation, as introduced in Section 2.1.1.

Support vector machine: an SVM regression model with an SE kernel.

For these models we use grid search to optimise hyperparameters. The grid search procedure uses 3-fold cross-validation *within the training set*, using two folds for training and one fold as a development set. Hyperparameter values are selected by averaging the best results obtained for each fold. We use scikit-learn (Pedregosa et al., 2011) as our underlying implementation. The hyperparameter grid for each model is defined as below:

Ridge	
λ (regularisation coefficient)	$[0.01, 0.1, 1, 10, 100]$
SVM	
C (error penalty)	$[0.01, 0.1, 1, 10, 100]$
ϵ (margin size)	$[0.001, 0.01, 0.1, 1, 10]$
ℓ (SE kernel lengthscale)	$[0.01, 0.1, 1, 10, 100]$

4.3.2 Results and analysis

Table 4.2 shows the obtained results averaged over all emotions. The proposed GP VSK outperform the linear models in all metrics, confirming our hypothesis. This is an encouraging result since it provides evidence that the string kernels are capturing information which is not available when using a simple averaging approach.

In contrast, our approach does not outperform the non-linear models. This shows that non-linearity is crucial for this task and motivates future extensions of the VSK in that direction. Between these models, the Matèrn kernels perform the best, showing that the simple smoothness assumption from the SE models (as pointed out in Section 2.3.3) is too restrictive for this task. This is an interesting finding since most previous work in kernel methods for text regression use the SE kernel as a black-box. Our results show that gains can be made by simply choosing a more appropriate non-linear kernel.

	NLPD ↓	MAE ↓	r ↑
Ridge	–	11.01	0.543
GP Linear	4.09	11.03	0.539
GP VSK	4.06	10.53	0.586
SVM SE	–	10.12	0.613
GP SE	4.03	10.09	0.611
GP Matèrn $3/2$	4.00	9.81	0.635
GP Matèrn $5/2$	4.01	9.88	0.628

Table 4.2 Results on the Affective Text dataset, averaged over all emotions and cross-validation folds.

On Table 4.3 we show the results obtained per emotion. The overall trend follows the results from Table 4.2, except for the emotion *surprise*. For this emotion, the VSK model obtained the largest gains in Pearson's r over the linear baselines and was able to even outperform the SE models. Surprise is the most common emotion in this dataset in terms of non-zero values. The fact it has the lowest correlation scores show that the main challenge in predicting this emotion is to be able to distinguish emotional values in a more fine-grained way, which is less captured by the aggregate metrics. In this sense, the performance of the VSK is a very interesting finding since it managed to capture subtle differences for the response variables better than even some of the non-linear methods.

4.3.3 Inspecting hyperparameters

We can probe into the obtained VSK hyperparameter values after optimisation to obtain insights about what kind of representation the kernel is learning. On Table 4.4 we show the median values for the hyperparameters across the 10 folds in the cross-validation procedure. First, we can see that λ_g has a very low value, meaning that gappy sequences are not particularly important for this task. In other words, all signal is captured by contiguous word sequences. The $\boldsymbol{\mu}$ values show a preference for ngrams up to 3 words, while longer sequences do not add much information.

The main conclusion we obtain from inspecting the hyperparameters is that most of the signal is captured by contiguous sequences up to 3 words. The fact that this representation is preferable for this task might indicate that it is possible to devise models with explicit sequences as features, instead of relying on a string kernel. However, this is not always obvious when devising a model for a new task. Here, the power of the string kernels shows

	Anger			**Disgust**			**Fear**		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Ridge	–	8.42	0.587	–	7.03	0.460	–	11.47	0.680
GP Linear	3.90	8.39	0.588	3.77	6.92	0.464	4.16	11.48	0.679
GP VSK	3.87	8.10	0.628	3.75	6.66	0.508	4.14	10.90	0.705
SVM SE	–	7.71	0.664	–	6.51	0.545	–	10.05	0.741
GP SE	3.84	7.61	0.662	3.73	6.34	0.546	4.08	10.17	0.739
GP Matèrn $3/2$	3.80	7.37	0.688	3.68	6.10	0.573	4.09	10.06	0.746
GP Matèrn $5/2$	3.82	7.44	0.680	3.70	6.14	0.558	4.08	10.06	0.745
	Joy			**Sadness**			**Surprise**		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Ridge	–	14.600	0.556	–	13.701	0.638	–	10.811	0.337
GP Linear	4.350	14.931	0.531	4.301	13.782	0.633	4.055	10.669	0.336
GP VSK	4.320	14.444	0.569	4.245	12.787	0.683	4.012	10.314	0.423
SVM SE	–	14.014	0.613	–	12.118	0.710	–	10.323	0.404
GP SE	4.277	13.675	0.607	4.220	12.421	0.705	4.022	10.344	0.409
GP Matèrn $3/2$	4.237	13.252	0.636	4.192	11.964	0.722	4.005	10.130	0.445
GP Matèrn $5/2$	4.251	13.361	0.630	4.205	12.104	0.718	4.010	10.179	0.436

Table 4.3 Per-emotion results on the Affective Text dataset. Each number is averaged over the 10 folds in the cross-validation procedure.

λ_g	7.36×10^{-7}
λ_m	0.0918
μ_1	12.37
μ_2	33.73
μ_3	154.51
μ_4	2.58
μ_5	8.54

Table 4.4 Median VSK hyperparameter values for the *surprise* emotion. λ_g and λ_m correspond to gap and match decay hyperparameters, respectively. The μ_n hyperparameters correspond to the n-gram size.

that it is able to find a simpler representation for this problem, even though it has the capacity to use more complex ones.

4.4 Joint Modelling using Multi-task Gaussian Processes

Until now we assumed that predicting each emotion is a separate task. However, the nature of this problem shows that one is actually interested in predicting *all* emotions from a natural language sentence. This is reflected in most datasets available for this task, where multiple

emotion annotations are available for each text fragment. In light of this, it makes sense to aim for a *joint* model, which is able to predict a *vector* of emotional values.

If we consider each emotion as a task, what we pursue is essentially *multi-task learning* (Caruana, 1997). The main benefit of such approach is that it allows to incorporate existing correlations (and maybe anti-correlations) between the tasks, which makes sense for Emotion Analysis. For instance, a sentence with a high *joy* value is very likely to have a low *sadness* value and vice-versa. By taking into account these interactions we can make better use of the available annotations.

In GP regression, multi-task learning is obtained by employing vector-valued GPs, which are sometimes referred as *multi-task GPs* (Bonilla et al., 2008). The main idea is to use the Kronecker product property of kernels (see Section 2.3.5) to define a GP with a combination of kernels, one between inputs and another one between tasks. In this Section, we propose to employ this idea to obtain a joint model over all emotions. This can not only enhance the predictive performance but also can bring insights about how tasks relate to each other by inspecting the obtained hyperparameters of this model.

4.4.1 Multi-task Gaussian Processes

The main idea behind multi-task GPs is to extend the standard GP regression model to vector-valued outputs. This is done by extending the model Gram matrix with the Kronecker product between the resulting Gram matrix on the inputs and a matrix $\mathbf{B}$ with dimensionality $T \times T$, where T is the number of dimensions or tasks. In the literature, this is called *Intrinsic Coregionalisation Model* (ICM) (Álvarez et al., 2012) and $\mathbf{B}$ is referred to as the *coregionalisation matrix*:

$$k_{\text{ICM}}(\mathbf{x}, \mathbf{x}') = \mathbf{B} \otimes k(\mathbf{x}, \mathbf{x}'). \tag{4.1}$$

The purpose of the matrix $\mathbf{B}$ is to model task covariances. The main goal is to learn $\mathbf{B}$ from data and this is done by including it in the set of model hyperparameters to be optimised via empirical Bayes. However, we need to ensure $\mathbf{B}$ is symmetric and positive semi-definite, otherwise k_{ICM} will not be a valid kernel. We can ensure these properties by parameterising the matrix using Probabilistic Principal Component Analysis:

$$\mathbf{B} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}^T + \text{diag}(\boldsymbol{\alpha}), \tag{4.2}$$

where $\tilde{\mathbf{L}}$ is an $T \times R$ matrix where R is the rank of the matrix $\mathbf{B}$ and $\boldsymbol{\alpha}$ is a vector of noise hyperparameters. The rank can be understood as the capacity of the manifold with

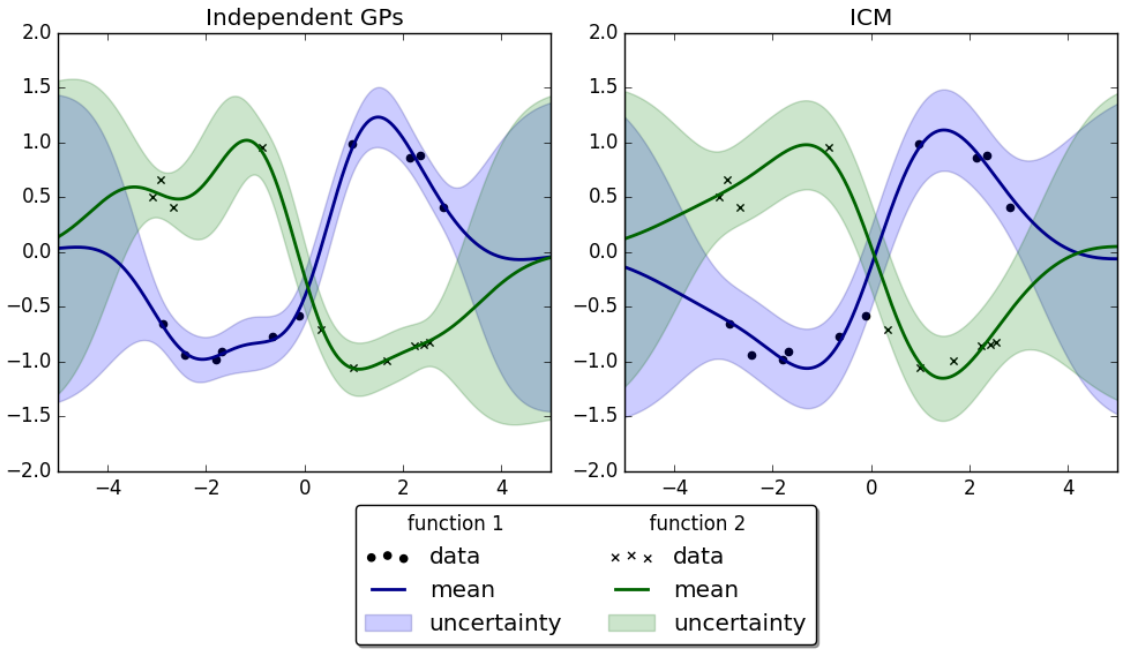


Fig. 4.1 Two correlated datasets modelled by a set of GP models. On the left we fit two independent GPs for each task and on the right we use a single GP within an ICM model. All kernels are SE and hyperparameters were optimised.

which we model the T tasks, while the vector $\boldsymbol{\alpha}$ allows for each task to behave more or less independently with respect to the global task. A higher rank for $\tilde{\mathbf{L}}$ allows more flexibility in modelling task correlations at the expense of increasing the number of hyperparameters to be optimised.

On Figure 4.1 we sample a set of points from a noisy cubic sine function (same one used in Section 2.3.1) and another set from the same function but with signal reversed. We then fit two independent GPs on the left, one for each dataset, and a single ICM model on the right, with both datasets. It is clear that the ICM can reach a better fit since it allows to explore the existing correlations between the two datasets. Specifically, the independent models have tighter uncertainty intervals in regions around the data points which quickly increase when extrapolating beyond these regions. Their mean curves also show an interpolating behaviour, especially in the regions where x is negative. These pathologies are evidence that the independent models are overfitting the data while the ICM avoids those completely.

We can inspect the optimised coregionalisation matrix to check what kind of correlations the model learned. For the ICM model of Figure 4.1, we get

$$\mathbf{B} = \begin{bmatrix} 9.067 & -8.820 \\ -8.820 & 8.715 \end{bmatrix}. \tag{4.3}$$

Note that the off-diagonal elements are negative, reflecting the anti-correlated behaviour between the two functions. In general, values in $\mathbf{B}$ are not equivalent to correlation scores since their optimised values depend on other model hyperparameters. Nevertheless, inspecting $\mathbf{B}$ can give insights in terms of finding functions or tasks which exhibit similar or reversed trends.

4.4.2 Experimental settings

Our main approach uses an ICM model with a range of different ranks for the coregionalisation matrix $\mathbf{B}$. For the kernel between inputs, we use the Matèrn $3/2$ kernel with averaged embeddings, as this was the kernel with the best obtained results in the single-task experiments. All kernels have an added bias term to account the response variable scaling.

We also employ a "mixed-noise" Gaussian likelihood, where each task has a different value for the Gaussian noise. While it is not an integral part of the ICM, it is commonly used in practice in multi-task GPs, as it tends to give better performance compared to use a single noise value for all tasks. This can easily be implemented by considering the noise can be represented as a kernel with diagonal values only, as explained in Section 2.3.5.

The baseline is composed of a set of single-task models using the same Matèrn $3/2$ kernel used in the ICM models. In other words, the same model we used as one of the non-linear baselines in Section 4.3.

4.4.3 Results and analysis

Table 4.5 shows averaged results for all emotions. The ICM models performed worse in terms of aggregate metrics but better in terms or correlation scores. The increase in NLPD could be due to worse uncertainty estimates but since MAE also increased we can conclude that the main cause is on worse point estimate predictions. On the other hand, the increase in correlation shows some evidence of better emotion discrimination within the test set instances.

	NLPD ↓	MAE ↓	r ↑
Single-task GP	4.00	9.81	0.635
ICM GP (rank 1)	4.02	9.85	0.640
ICM GP (rank 2)	4.02	9.85	0.640
ICM GP (rank 3)	4.03	9.85	0.639
ICM GP (rank 4)	4.03	9.84	0.639
ICM GP (rank 5)	4.03	9.84	0.639

Table 4.5 Multi-task results on the Affective Text dataset, averaged over all emotions and cross-validation folds.

	Anger			**Disgust**			**Fear**		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Single-task GP	3.80	7.37	0.688	3.68	6.10	0.573	4.09	10.06	0.746
ICM GP (rank 1)	3.82	7.46	0.686	3.70	6.20	0.578	4.10	10.05	0.747
ICM GP (rank 2)	3.82	7.45	0.687	3.71	6.19	0.578	4.09	10.05	0.748
ICM GP (rank 3)	3.82	7.43	0.688	3.72	6.20	0.577	4.09	10.05	0.748
ICM GP (rank 4)	3.83	7.44	0.687	3.72	6.20	0.577	4.09	10.04	0.748
ICM GP (rank 5)	3.82	7.44	0.687	3.71	6.20	0.577	4.09	10.04	0.748
	Joy			**Sadness**			**Surprise**		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Single-task GP	4.24	13.25	0.636	4.19	11.96	0.722	4.01	10.13	0.445
ICM GP (rank 1)	4.30	13.34	0.637	4.23	12.00	0.722	3.99	10.04	0.466
ICM GP (rank 2)	4.30	13.34	0.638	4.23	12.00	0.722	3.99	10.06	0.466
ICM GP (rank 3)	4.30	13.33	0.638	4.23	12.00	0.722	3.99	10.06	0.465
ICM GP (rank 4)	4.31	13.33	0.637	4.23	12.00	0.723	3.99	10.06	0.464
ICM GP (rank 5)	4.31	13.32	0.638	4.23	11.99	0.723	3.99	10.07	0.463

Table 4.6 Per-emotion results for multi-task models on the Affective Text dataset. Each number is averaged over the 10 folds in the cross-validation procedure.

In terms of different ranks for the coregionalisation matrix, no particular difference was caught between the models. This means that the interactions between the tasks can be captured by low rank matrices.

On Table 4.6 we show per-emotion results. Interestingly, we again see most gains in the *surprise* emotion, especially on correlation scores. It is also interesting that correlation was mostly not affected for the other emotions as well. On the other hand, aggregate metrics follow the same trend as in Table 4.5, except for *surprise*.

An important aspect of the ICM is that it uses the same kernel in inputs for all tasks. This is in contrast with the single-task models, which can have emotion-specific kernels with their own fine-tuned hyperparameter values. This loss in capacity might be the reason why

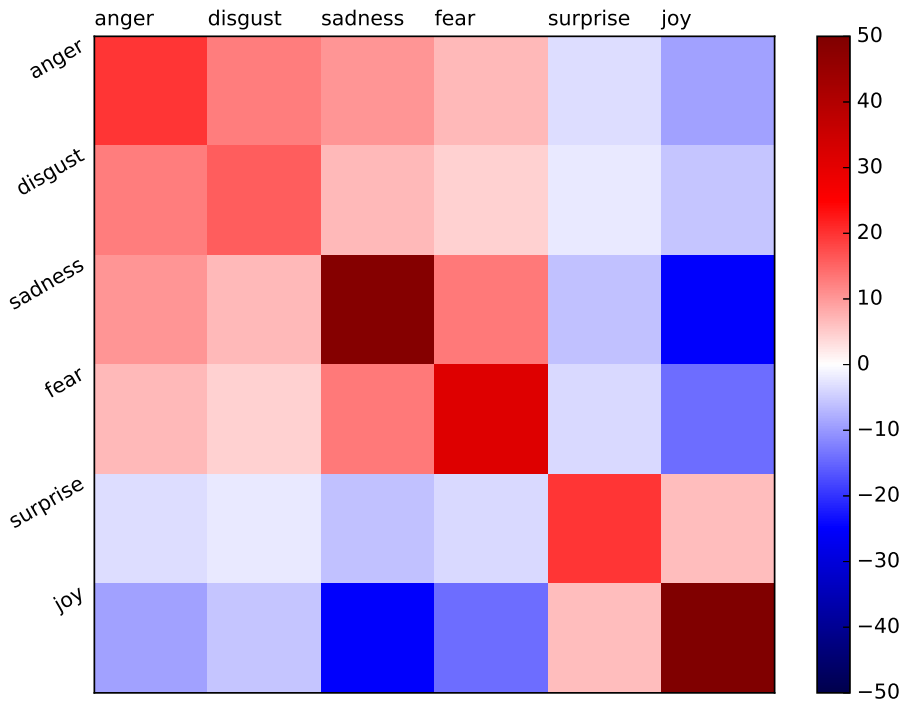


Fig. 4.2 Heatmap of a coregionalisation matrix obtained from a rank 2 model. Warm colors show positive values and cool colors represent negative values. Emotions were reordered to clarify the trends we obtained.

aggregate metrics showed worse performance compared to the single-task models. On the other hand, the ICM adds task correlations through the coregionalisation matrix, which in turn might explain the slightly better correlation scores.

4.4.4 Inspecting hyperparameters

An additional benefit of the ICM models is that the optimised coregionalisation matrix is interpretable. We can probe the matrix to obtain insights about how tasks correlate with each other. Figure 4.2 shows an example of a learned coregionalisation matrix for a rank 2 model, in a heatmap format. The diagonal terms show how much a task benefit from signal from the others: the higher the value, the more "independent" it is, i.e., it relies less on the other tasks. The off-diagonal terms show inter-task covariances, which explain possible interactions between tasks.

The first behaviour we can notice from the matrix is that tasks are grouped in two clusters. The top left one is composed of mostly "negative" emotions, while the lower right groups *joy* and *surprise* together. This is an interesting finding because while expect that *surprise* to

be more neutral with respect to the other emotions it actually have a degree of correlation with *joy*, a "positive" emotion. Another peculiar finding is that *fear* and *anger* show positive correlation, which contrasts with some emotion taxonomies that treats these two emotions as being opposites (Mohammad and Turney, 2011; Plutchik, 1980).

Looking at the diagonal values, we can see that *disgust* seems to benefit the most from other tasks' emotional labels, which is expected since it is the least frequent emotion in terms of non-zero values. Conversely, *joy* and *sadness* seem to be the most "independent" emotions, in the sense they do not use much of the signal from the other emotions. They also show the highest degree of anti-correlation, as expected.

4.5 Discussion

In this Chapter we introduced Gaussian Process regression models for Emotion Analysis. We focused on predicting emotions from news headlines using the taxonomy proposed by Ekman (1993). Our goals consisted of obtaining better models through the use of improved text representations and by employing joint models which treat emotion prediction as a multi-task learning problem.

For the first goal we proposed the use of string kernels to obtain better text representations. Our experiments resulted in better performance compared to linear models based on averaged word embeddings, showing that string kernels can indeed capture more linguistic information using the same set of resources. We also showed how inspecting optimised hyperparameters can give insights on what kind of representation the string kernels are learning. A drawback of our approach is that it still results in a linear model, which might explain why it was outperformed by models that incorporate non-linear behaviour. This points to a promising research direction, namely extending string kernels through the use of non-linear patterns.

The results obtained for our second goal were less conclusive in terms of predictive performance. While the multi-task models tend to give better correlation in the predictive response variables, their performance with respect to aggregate metrics were not as good as in the single-task models. An inherent restriction of our approach is that function behaviour is addressed by a single input kernel, while single-task models can have different, fine-tuned input kernels. Therefore, one possible extension is to incorporate multiple kernels in the multi-task model, which has been proposed in previous work (Álvarez et al., 2012). This can prove challenging though, since it increases the number of hyperparameters, making optimisation more difficult and prone to overfitting.

While the multi-task model we proposed gave mixed results in predictive performance it nevertheless has the additional benefit of generating a matrix that encodes task covariances. Here we showed how we can probe into this matrix to obtain insights about how emotions relate to each other. We saw that while some (anti-)correlations are expected (like *joy/sadness*), others contradict some proposed taxonomies (the case of *anger/fear*). These are very interesting findings that have the potential to lead to new directions in general studies of emotion.

Overall, the Emotion Analysis benchmarks showed evidence that our Gaussian Process approach achieves the main goals of this thesis. The flexibility provided by GPs lets us employ a range of different kernels and extend the model to multi-task learning setting. This allowed us to reach better uncertainty estimates, as measured by NLPD, and also increase model interpretability. Furthermore, we also obtained promising results on the combination of GPs and string kernels in a real world task, which showed that going beyond simple text representations can be beneficial. In the next Chapter we benchmark our approach on a second real world task, Machine Translation Quality Estimation, where we further explore uncertainty estimation aspects and test models based on combining GPs with tree kernels.

Chapter 5

Machine Translation Quality Estimation

Machine Translation (MT) is the task of automatically translate texts from a source natural language to a target natural language (Jurafsky and Martin, 2000). Before the 1990's most MT systems were designed as sets of translation rules which were manually encoded by experts. This paradigm started to shift to machine learning approaches with the seminal work of Brown et al. (1990), which proposed to automatically learn translation rules from parallel corpora, aggregates of texts available in multiple languages. This idea then matured into what we known as Statistical Machine Translation (SMT) (Koehn, 2010), which has dominated MT research for more than 20 years and it is still widely used in commercial applications.

The revival of neural networks in the last 5 years led to another paradigm shift in MT, giving rise to the field of Neural Machine Translation (NMT). While it is still based on machine learning and parallel corpora, it has fundamental differences from SMT, the main one being the assumption that words can be represented in a continuous space through the use of embeddings. SMT, on the other hand, is based on symbolic probabilistic models over the words (or other text segments) themselves. Proposed architectures for NMT are usually based on recurrent NN components (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014), and the state-of-the-art is based on the so-called attention models (Bahdanau et al., 2015; Luong et al., 2015).

Despite these recent breakthroughs, MT is far from solved. This is not only due to errors introduced by MT systems but also because generally measuring translation quality is an open problem which depends on a number of factors. For example, a translation that is good enough for gisting purposes might not be acceptable for printing in a technical manual. Even when the end task is known in advance, people might disagree on the best translation for a text segment due to varying level of expertise, style differences or personal preferences. On the other hand, empirical evidence shows there are benefits in using MT systems in real world

scenarios, such as enhancing productivity via post-editing, for instance (Plitt and Masselot, 2010). In such scenarios, having a reliable quality measure of translated segments output can further improve the usability of MT systems.

Assessing the quality of new, unseen machine translated texts is the topic of Quality Estimation (QE) (Blatz et al., 2004; Specia et al., 2009b). Unlike traditional MT evaluation techniques, which are based on comparing system outputs to references, QE does not assume the existence of such references. The aim is to provide quality indicators that usually refer to some specific end task. Applications include filtering machine translated sentences that would require more post-editing effort than translation from scratch (Specia et al., 2009a), selecting the best translation from different MT systems (Specia et al., 2010) or between an MT system and a translation memory (He et al., 2010), and highlighting segments that need revision (Bach et al., 2011).

In this Chapter we focus on QE post-editing scenarios. More specifically, we build models that aim at predicting the post-editing time per word in a translated segment or *post-editing rate* (PER, as coined by Graham (2015)) as a quality indicator. This decision was based on the fact that time is a more complete measure of post-editing effort, capturing not only technical but cognitive effort (Koponen et al., 2012). Additionally, time is more directly applicable in real translation environments, where uncertainty estimates could be useful as they relate directly to productivity measures.

Table 5.1 shows some examples of QE instances present in the English-Spanish dataset we use in our benchmarks. Notice that even when no post-edition was made (the example in the top block) the annotation procedure still logged a small amount of post-editing time. This is expected as the measure should also consider the time the expert used to inspect the quality of the MT output, which in this case was already acceptable. The other two blocks are examples of low and high PER when the MT output was edited. The middle block shows an instance where simple post-editing actions (mostly pronoun deletion) were made, while in the bottom one the MT system output was of very low quality and the annotator had to spend a significant amount of time fixing it.

We give an overview of related work in Section 5.1 and then proceed to show our contributions with respect to the task. In Section 5.2 we describe the datasets we use in our experiments and detail the set of hand-crafted features which are employed in our models. Section 5.3 addresses the topic of uncertainty estimates in QE, where we introduce Warped Gaussian Processes and employ them for QE applications in asymmetric risk scenarios and active learning. In Section 5.4 we propose to move away from models based on hand-crafted

Source: Phil T. Pulaski, the department's chief of detectives, was also there. **MT output:** Phil T. Pulaski, el jefe del departamento de detectives, también estaba allí. **Post-edition:** Phil T. Pulaski, el jefe del departamento de detectives, también estaba allí. **Post-editing rate:** 0.23 seconds/word **Post-editing time:** 3.448 seconds
Source: And you're going to get through this, and you're going to recover and you're going to come back stronger than ever. **MT output:** Y usted va a salir de esta, y usted va a recuperarse y usted va a volver más fuerte que nunca. **Post-edition:** Y vas a salir de esta, y vas a recuperarse y vas a volver más fuerte que nunca. **Post-editing rate:** 0.576 seconds/word **Post-editing time:** 13.254 seconds
Source: Tales of cheating on school and college tests are rife. **MT output:** Las historias de fraudes en la escuela y colegio pruebas son moneda corriente. **Post-edition:** Hacer trampas en exámenes de la escuela y el colegio es corriente. **Post-editing rate:** 7.95 seconds/word **Post-editing time:** 111.301 seconds

Table 5.1 Quality Estimation instances, taken from the English-Spanish dataset used in our experiments. The top block shows an instance where no post-edition was made. In the middle block we give an instance with low PER, but with some post-editions. The bottom block shows an instance with high PER.

features by using tree kernels over the syntactic structures of both source and translated segments. We wrap up with a discussion in Section 5.5.

5.1 Related Work

Quality Estimation is a fairly recent topic in NLP. Blatz et al. (2004) is to our knowledge the first comprehensive work to address this problem. In their setting, reference-based MT evaluation metrics such as NIST (Doddington, 2002) are employed as quality metrics to predict. The problem is addressed as binary classification: sentences which score above a threshold are considered of acceptable quality. Their main finding is that they are able to achieve better performance by employing QE as a separate layer instead of just relying on the underlying MT system probability scores (their baseline). This is due to the fact that this layer can be any kind of ML model, which allows the use of an arbitrary number of features that can correlate to MT quality. This finding paved the way for arguing on the use of QE as a model on its own, instead of trying to incorporate it as a component in MT systems.

Quirk (2004) proposed to use manually annotated labels as a quality metric in the form of a Likert scale (integers) in the $[1, 4]$ interval. His main finding is that training models on even a small quantity of manually annotated data can outperform models trained on large data with labels obtained automatically from MT metrics. This shaped the topic to move away from using MT evaluation metrics as quality indicators and towards studying better human-based indicators in the form of manual annotations.

Specia et al. (2009a) first addressed QE as regression instead of threshold-based binary classification. Quality indicators are either NIST or Likert scores but the goal is to accurately predict the indicator itself instead of finding a decision boundary between good and bad translations. They also introduced the concept of *glass-box* features, which are obtained by inspecting the underlying MT system used to obtain the translations, as opposed to *black-box* which are agnostic to the system. Their main finding is that most discriminative features are actually black-box: glass-box ones do not add much to QE models. This further motivated the task to be modelled as a separate layer, since black-box features do not require knowledge about the provenance of the translated segments.

Implicit quality metrics were first proposed by Specia and Farzindar (2010), which use Human Translation Error Rate (HTER) (Snover et al., 2006) as a quality indicator. HTER is based on post-editing the output of an MT system, where a human expert is asked to modify a translated segment until it reaches an acceptable quality. The metric is then obtained by counting the number of editing operations made by the expert and normalising it by the segment length. This setting was also used by Specia (2011), while also proposing the use of post-editing *time* as an indicator. Her findings show that both HTER and post-editing time correlate with explicit Likert indicators. This shows that accurate QE indicators can be obtained in a transparent, task-specific manner.

All the findings cited above shaped the task as being addressed through independent, feature-based regression models that predict task-specific quality indicators, which in turn usually come from post-editing environments. While this approach has been shown to be the most useful it also raises the concern of data scarcity. In this setting, we ideally would like to have separate QE models for each possible scenario, including specific expert translators, language pairs, MT systems and text domains. Recent work tried to tackle these issues through the use of multi-task models (Cohn and Specia, 2013; de Souza et al., 2014). Another body of work aims at treating QE as an online learning model, which updates its parameters as the expert translates new segments and, in theory, is able to give better results as it progresses through time (de Souza et al., 2015; Turchi et al., 2014).

Many advances in the field were pushed by the shared tasks annually run at the Workshop in Machine Translation (WMT) (Bojar et al., 2013; Callison-Burch et al., 2012, *inter alia)*. These shared tasks provided new datasets in a variety of language pairs, mostly annotated with post-editing based metrics. They were very successful in proposing new features for QE, many of which were implemented in QuEst (Specia et al., 2013), a QE framework focused on feature extraction and model training.

Despite these advances, the task is still focused on accurate prediction of quality metrics through feature-based models. However, in real world environments quality indicators are never the end goal but a way to present information for decision making, such as filtering for instance. In this sense, we argue that uncertainty estimates should be provided as additional information for decision making. Furthermore, while hand-written features can be very useful, they require feature engineering, which can be costly and time consuming. Here we propose solutions for these problems using the models we described in Chapters 2 and 3 and assess them in a series of benchmarks.

5.2 Datasets and Features

Our experiments comprise datasets in three language pairs which are annotated with post-editing time measurements at the sentence level:

English-Spanish (en-es) This dataset was used in the WMT14 QE shared task (Bojar et al., 2014). It contains 858 sentences translated by one MT system and post-edited by a professional translator.

French-English (fr-en) Described in Specia (2011), this dataset contains 2,525 sentences translated by one MT system and post-edited by a professional translator. We filtered the dataset by removing instances where post-editing time is zero, resulting in 2,497 sentence pairs in total.

English-German (en-de) This dataset is part of the WMT16 QE shared task[1], which contains sentences translated by a single MT system and post-edited by 6 professional translators. We chose a subset containing translations from a single expert, totalling 2,828 instances.

We obtain post-editing rates by normalising these measurements with respect to the length of the translated sentence and use these values as our response variables.

[1] www.statmt.org/wmt16

While state-of-the-art systems use a wide range of feature sets, these are mostly resource-intensive and language-dependent, and therefore not equally applicable to all our language pairs. Therefore we employ a set of 17 black-box features which are part of the baseline systems used in the WMT QE shared tasks. These features were initially proposed by Specia et al. (2009a) as a compromise between discriminative power and robustness across language pairs and in terms of resource availability. While considered a baseline, it is considered a strong one in terms of predictive performance, commonly outperforming shared task submissions.

The feature set is described below. We split the description between source and target sentence-based features:

- Source features:
  1. Sentence length (in words);
  2. Average word length (in characters);
  3. Number of punctuation marks;
  4. Language model probability;
  5. Average number of translations per word, where $p(t|s) > 0.2$;
  6. Average number of translations per word, where $p(t|s) > 0.01 \times \mathrm{IF}(s)$, where IF is the inverse frequency of the word in a corpus;
  7. Percentage of words present in a corpus;
  8. Percentage of words present in the $< 25\%$ frequency quartile of a corpus;
  9. Percentage of bigrams present in the $< 25\%$ frequency quartile of a corpus;
  10. Percentage of trigrams present in the $< 25\%$ frequency quartile of a corpus;
  11. Percentage of words present in the $> 75\%$ frequency quartile of a corpus;
  12. Percentage of bigrams present in the $> 75\%$ frequency quartile of a corpus;
  13. Percentage of trigrams present in the $> 75\%$ frequency quartile of a corpus;
- Target features:
  14. Sentence length (in words);
  15. Average number of word type occurrences in the sentence;
  16. Number of punctuation marks;

17. Language model probability.

We use QuEst to extract these features for all sentence pairs. For the *en-es* and *fr-en* pairs, the corpus used for the language models and the frequency-based features is a combination of Europarl (Koehn, 2005) and news corpora. Features 5 and 6 were obtained from an IBM Alignment Model 1, trained the same combined corpus. For the *en-de* dataset we use an IT domain corpus for all related features.

5.3 Uncertainty Estimates for Quality Estimation

As explained in Section 5.1, QE models are mainly focused on providing accurate predictions for quality indicators. However, QE is usually a hard task, especially considering the huge domain space and the subsequent data scarcity problem. For instance, the best system in the latest sentence-level QE shared task (Bojar et al., 2016) achieved 0.525 Pearson's correlation, showing that we are still far from solving the task. This means that even predictions from the state-of-the-art models can not be considered fully reliable for decision making.

In light of this, we argue that QE models should provide uncertainty estimates for predictions in order to be useful in real world scenarios. Consider for example a post-editing scenario where professional translators use MT in an effort to speed-up the translation process. A QE model can be used to determine if an MT segment is good enough for post-editing or should be discarded and translated from scratch. But since QE models are not perfect they can end up allowing bad MT segments to go through for post-editing because of a prediction error. In such a scenario, having an uncertainty estimate for the prediction can provide additional information for the filtering decision. For instance, in order to ensure good user experience for the human translator and maximise translation productivity, an MT segment could be forwarded for post-editing only if a QE model assigns a high quality score with *low uncertainty* (high confidence). Such a decision process is not possible with point estimates only.

In this Section we show how Gaussian Processes can be employed to obtain good uncertainty estimates from well-calibrated predictive distributions for quality indicators. More specifically, we use Warped GPs to build QE models since they can address the non-Gaussian behaviour of quality metrics (in our case, post-editing rates). Finally, we also show two applications for predictive distributions, one based on asymmetric risk scenarios and another one in an active learning setting.

5.3.1 Warped Gaussian Processes

In a standard GP regression model the response variables are modelled using a Gaussian, which has support over $\mathbb{R}$. However, in real applications sometimes the variables do not follow this assumption. For example, time measurements are strictly positive: modelling these values as Gaussian can result in predictive distributions that assign mass to impossible values like negative time.

For strictly positive variables a common trick is to model the logarithm of the values, since this transformation maps positive values to the real line. However, there is no reason to believe the $\log$ is the best mapping since there is no guarantee the transformed response variables are Gaussian distributed. A better solution would be to learn the mapping from the training data.

Warped GPs (Snelson et al., 2004) are an extension of GPs that allows the learning of arbitrary mappings. It does that by placing a monotonic *warping function* over the observations and modelling the warped values inside a standard GP. The posterior distribution is obtained by applying a change of variables:

$$p(y|\mathbf{x}) = \frac{f'(y)}{\sqrt{2\pi\sigma^2}} \exp - \frac{1}{2}\left(\frac{f(y)-\mu}{\sigma}\right)^2,$$

where μ and σ^2 are the mean and standard deviation of the latent (warped) response variable and f and f' are the warping function and its gradient.

The warping function should be flexible enough to allow the learning of complex mappings but it needs to be monotonic. Snelson et al. (2004) proposes a parametric form composed of a sum of $\tanh$ functions, similar to a neural network layer:

$$f(y) = y + \sum_{i=1}^{I} a_i \tanh(b_i(y + c_i)),$$

where I is the number of $\tanh$ terms and $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$ are treated as model hyperparameters and optimised jointly with the kernel and likelihood hyperparameters. Large values for I allow more complex mappings to be learned but make optimisation more difficult and increase the risk of overfitting.

Figure 5.1 shows the difference between a standard GP and a Warped GP when fitted to the noisy cubic sine function introduced in Section 2.3.1 and showed in Figure 2.5. Although both models use an SE kernel, the Warped GP was able to find a much better fit to the data, due to its ability to model non-Gaussian response variables.

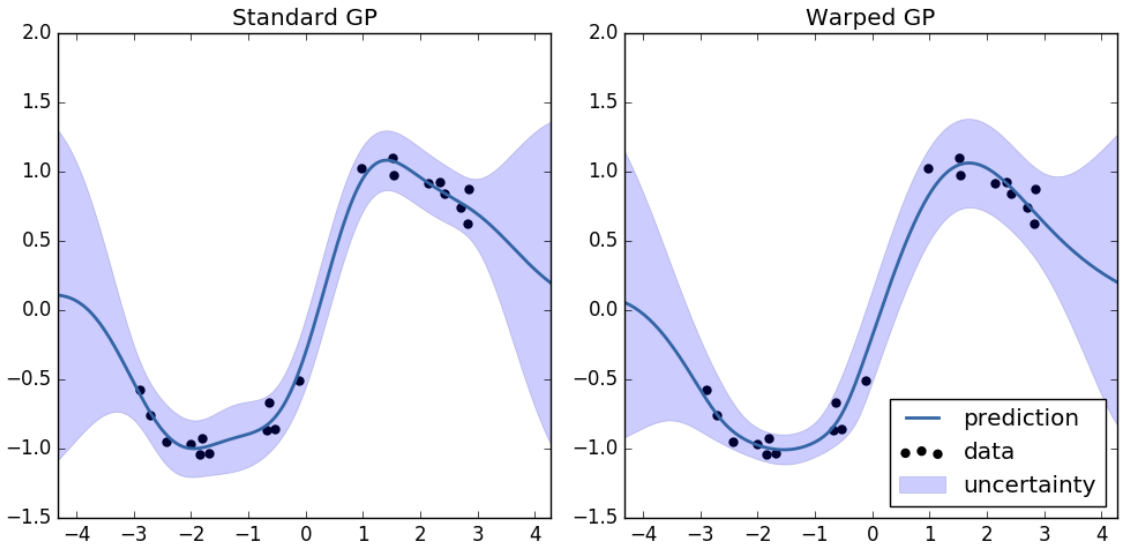


Fig. 5.1 A standard GP and a Warped GP fit to the noisy cubic sine function, after hyperparameter optimisation.

Another example is shown on Figure 5.2, where we model a noisy exponential function. We can see that the Warped GP is able to learn that the function is strictly positive by assigning most of its probability mass where the response variable is greater than zero. On the other hand the standard GP ends up assigning uncertainty to negative values due to the Gaussianity assumptions. The learned warped function is shown on Figure 5.3 and exhibits a logarithm trend, which is expected since the response variables were drawn from an exponential function.

Notice that warping functions do not change the *support* of the predictive distribution, meaning that a warped Gaussian will have mass over all possible values in the real line. In other words, they do not guarantee that the output will be within the desired range. Their role is to reshape the output distribution so that values outside the range become very unlikely (low probability).

Decision theory for Warped GPs

If there is interest in obtaining point estimate predictions from Warped GP models one can apply the same theoretical background we introduced in Section 2.4.1, namely predicting the median or the mean when minimizing absolute or squared error, respectively. However, unlike standard GPs, in Warped GPs the predictive distributions are in general not Gaussian

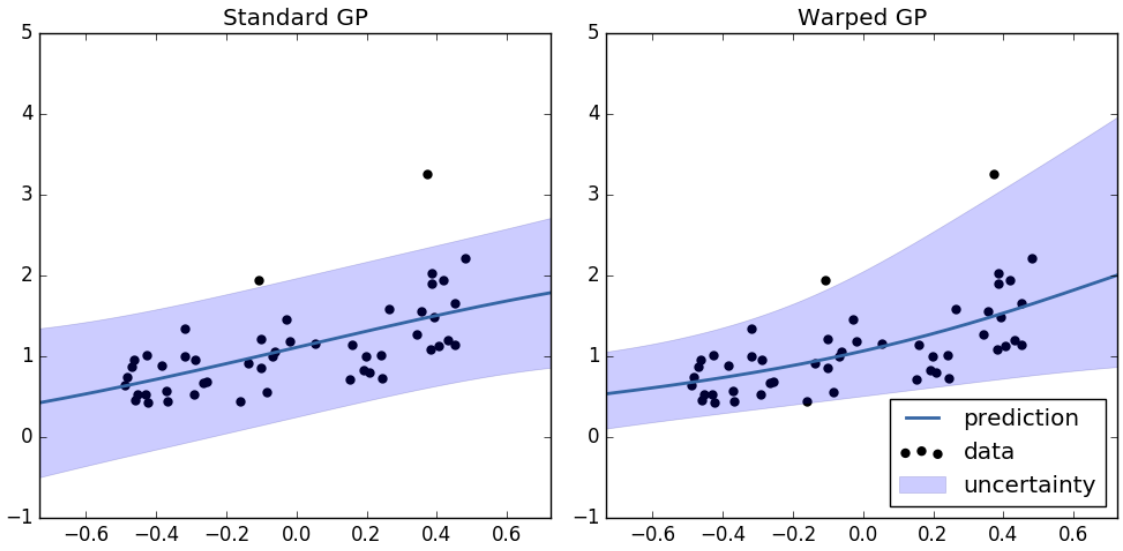


Fig. 5.2 A standard GP and a Warped GP fit to a noisy exponential function, after hyperparameter optimisation.

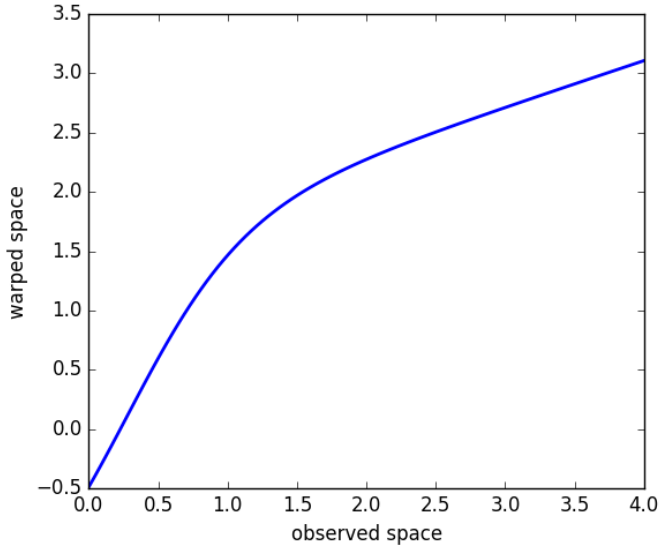


Fig. 5.3 The learned warping function after fitting a Warped GP to data drawn from a noisy exponential function.

so we need to obtain the median and the mean separately. For this, we can use the fact that distributions in the warped space are Gaussians to facilitate the calculations.

The median can obtained by simply pushing the warped space Gaussian median through the inverse of the warping function:

$$y_*^{\text{med}} = f^{-1}(\mu_*). \tag{5.1}$$

where we use the fact that the median and the mean are the same values in a Gaussian. While the inverse of the warping function is usually not available in closed form, we can use its gradient to have a numerical estimate. The mean is obtained by integrating y^* over the latent density:

$$\mathbb{E}[y_*] = \int f^{-1}(z)\mathcal{N}_z(\mu_*, \sigma_*^2)dz, \tag{5.2}$$

where z is the latent variable. This can be easily approximated using Gauss-Hermite quadrature since it is a one dimensional integral over a Gaussian density.

5.3.2 Experimental settings

Our hypothesis is that Warped GPs are able to better model uncertainty in QE compared to standard GPs. To test this, we train a set of different GP models using the datasets and features detailed in Section 5.2:

- Standard GP;
- Warped GP;
  - $\log$ warping function;
  - $\tanh$ warping function with 1 term ($\tanh 1$);
  - $\tanh$ warping function with 2 terms ($\tanh 2$);
  - $\tanh$ warping function with 3 terms ($\tanh 3$).

For each model we also employ three different kernels, squared exponential (SE), Matèrn $3/2$ and Matèrn $5/2$, totalling 15 experiments. Kernels use ARD, having one lengthscale parameter per feature (17 in total). We optimise hyperparameters using a two step process. First we tie all lengthscales, making the kernel isotropic, and maximise the marginal likelihood. Then we use the value found in the first step as a starting point for the final optimisation, where we untie the lengthscales. A similar procedure was performed by Cohn and Specia (2013), which has been shown to achieve better optima compared to a simple random restarts method.

	English-Spanish			French-English			English-German		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Standard GP									
SE	1.632	0.828	0.362	1.039	0.491	0.322	1.865	1.103	0.359
Matèrn $3/2$	1.649	0.862	0.330	1.040	0.491	0.320	1.861	1.098	0.369
Matèrn $5/2$	1.637	0.853	0.340	1.037	0.490	0.320	1.861	1.098	0.369
Warped GP $\log$									
SE	1.277	0.798	0.368	0.855	0.493	0.314	1.581	1.063	0.360
Matèrn $3/2$	1.271	0.793	0.380	0.857	0.493	0.310	1.580	1.063	0.363
Matèrn $5/2$	1.272	0.794	0.376	0.855	0.493	0.313	1.581	1.064	0.363
Warped GP $\tanh 1$									
SE	1.274	0.790	0.375	0.840	0.482	0.322	1.597	1.068	0.343
Matèrn $3/2$	1.272	0.790	0.379	0.840	0.482	0.317	1.593	1.064	0.351
Matèrn $5/2$	1.274	0.791	0.376	0.834	0.482	0.320	1.595	1.068	0.349
Warped GP $\tanh 2$									
SE	1.275	0.792	0.376	0.775	0.483	0.323	1.570	1.060	0.359
Matèrn $3/2$	1.281	0.791	0.382	0.754	0.483	0.313	1.570	1.060	0.362
Matèrn $5/2$	1.282	0.792	0.278	0.768	0.486	0.306	1.570	1.060	0.361
Warped GP $\tanh 3$									
SE	1.282	0.791	0.380	0.803	0.484	0.314	1.569	1.062	0.359
Matèrn $3/2$	1.282	0.791	0.380	0.760	0.486	0.302	1.567	1.058	0.364
Matèrn $5/2$	1.275	0.790	0.385	0.751	0.482	0.320	1.566	1.059	0.365

Table 5.2 Intrinsic evaluation results for all datasets. The first three rows correspond to standard GP models while the other rows show results for different Warped GP models.

We assess our models in terms of NLPD, MAE and Pearson's r, using a 10-fold cross-validation procedure. Our main interest is on NLPD since our goal is to provide good predictive distributions on unseen data. To evaluate MAE and Pearson's r we use the median of the distribution as a point prediction.

5.3.3 Intrinsic evaluation results

Table 5.2 shows the results obtained for all language pairs. The first column shows an interesting finding in terms of model learning: using a warping function drastically decreases NLPD. The main reason behind this is that standard GPs assign more probability mass over negative values compared to the Warped GP models..

In terms of different warping functions, using the parametric $\tanh$ function with 3 terms performs better than the $\log$ for the **fr-en** and **en-de** datasets. This is not the case of the **en-es** dataset, where the $\log$ function tends to perform better. We believe that this is due to the smaller dataset size. The gains from using a Matèrn kernel over SE are less conclusive. While they tend to perform better for **fr-en**, there does not seem to be any difference in the

other datasets. Different kernels can be more appropriate depending on the language pair, but more experiments are needed to verify this, which we leave for future work.

The differences in uncertainty modelling are by and large not captured by the point estimate metrics. While MAE does show gains from standard to Warped GPs, it does not reflect the difference found between warping functions for **fr-en**. Pearson's r is also quite inconclusive in this sense, except for some observed gains for **en-es**. This shows that NLPD indeed should be preferred as a evaluation metric when proper prediction uncertainty estimates are required by a QE model.

To obtain more insights about the performance in uncertainty modelling we inspected the predictive distributions for two sentence pairs in the **fr-en** dataset. We show the distributions for a standard GP and a Warped GP with a $\tanh 3$ function in Figure 5.4. In both cases the Warped GP was able to learn that labels are positive, putting negligible mass on negative numbers, unlike the Standard GPs. Also, in this example we can see that when the Warped GP makes a mistake in the prediction it pushes mass away from it, as evidenced by the higher peak around zero. On the other hand, the Standard GP distributions are very similar regardless of when it makes a good or bad prediction. Finally, notice that Warped GP distributions are multimodal in these particular examples.

Inspecting the resulting warping functions can bring additional modelling insights. In Figure 5.5 we show instances of $\tanh 3$ warping functions learned from the three datasets and compare them with the $\log$ warping function. We can see that the parametric $\tanh 3$ model is able to learn non-trivial mappings. For instance, in the **en-es** case the learned function is roughly logarithmic in the low scales but it switches to a linear mapping after $y = 4$. Notice also the difference in the scales, which means that the optimal model uses a latent Gaussian with larger variance.

Comparison with the state-of-the-art

The English-Spanish dataset was developed for a shared task at WMT14. Here we perform another set of experiments in this dataset, this time using the official training and test split. This way we can compare our results with other systems submitted for the shared task and put our methods into a broader perspective. We restrict our analysis to Warped GPs with a $\tanh 3$ warping function, since other similar methods performed comparably.

Table 1 show the results obtained with the official split, alongside other shared task participants. The Warped GP models consistently outperform the shared task baseline, an SVM regression model with an SE kernel. This is a very encouraging result since both our

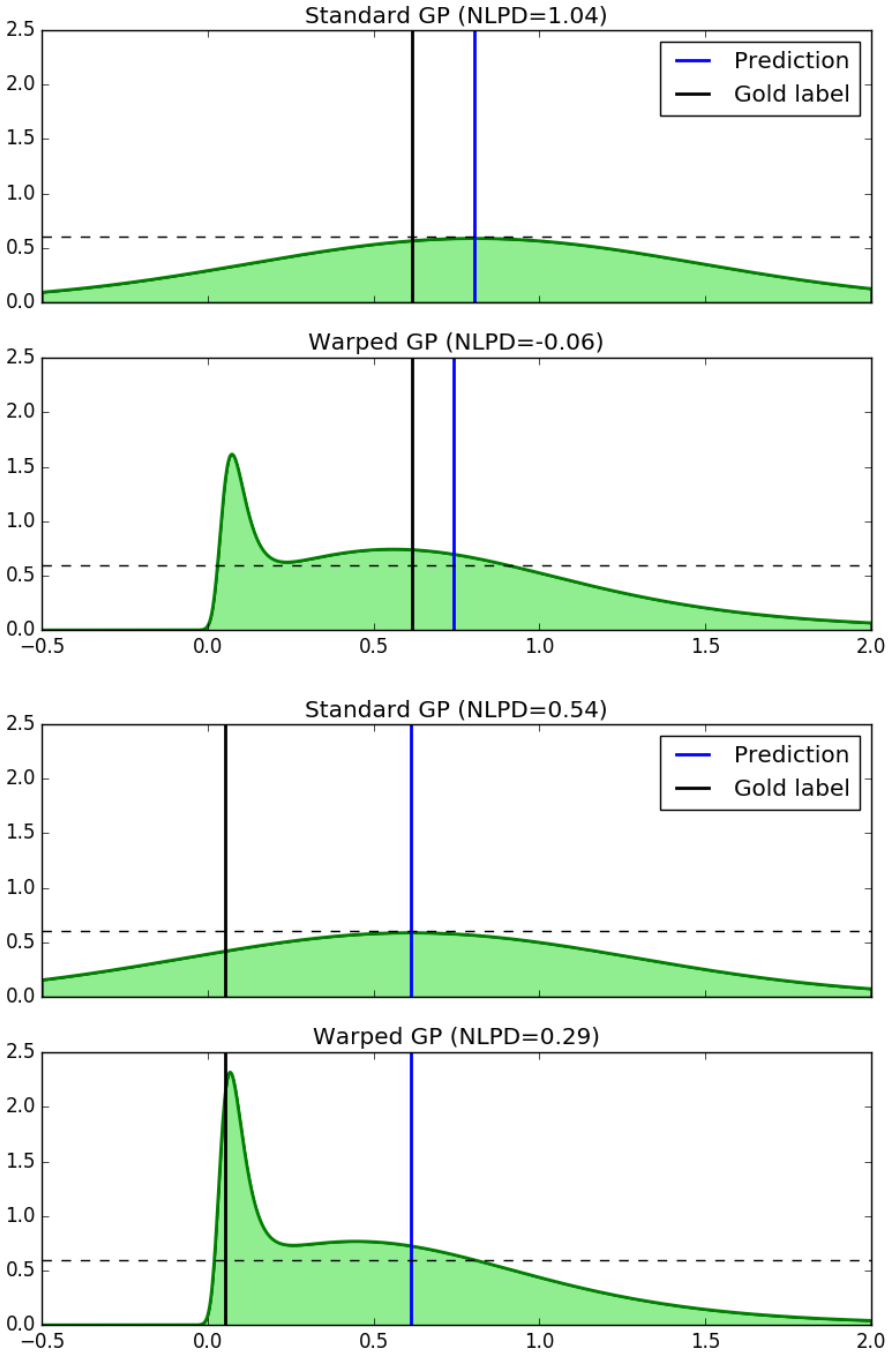


Fig. 5.4 Predictive distributions for two **fr-en** instances under a Standard GP and a Warped GP. The top two plots correspond to a prediction with low absolute error, while the bottom two plots show the behaviour when the absolute error is high. The dashed line ($y = 0.6$) is for visualisation purposes.

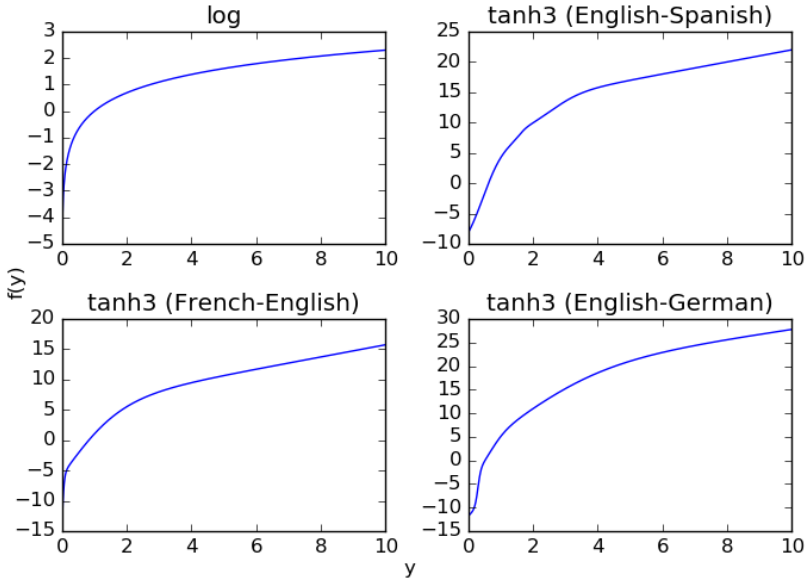


Fig. 5.5 Warping function instances from the three datasets. The vertical axes correspond to the latent warped values. The horizontal axes show the observed response variables, which are always positive in our case since they are post-editing rates.

	MAE ↓	RMSE ↓
RTM-DCU/RTM-SVR	16.77	26.17
MULTILIZER/MLZ2	17.07	25.83
SHEFF-lite	17.13	27.33
MULTILIZER/MLZ1	17.31	25.51
SHEFF-lite/sparse	17.42	27.35
FBK-UPV-UEDIN/WP	17.48	25.31
RTM-DCU/RTM-RR	17.50	25.97
FBK-UPV-UEDIN/NOWP	18.69	26.58
USHEFF	21.48	34.28
Baseline	21.49	34.28
Warped GP tanh3		
SE	17.86	26.74
Matèrn $3/2$	17.64	26.47
Matèrn $5/2$	17.90	26.78

Table 5.3 Results on the English-Spanish dataset, using the official shared task split. The results in the top block correspond to the shared task participants (obtained from Bojar et al. (2014)) while the bottom block contains our proposed models.

models and the baseline use the same features. Our models also are competitive with most of the submitted systems, even though they rely on richer feature sets.

5.3.4 Asymmetric risk scenarios

Evaluation metrics for QE, including those used in the WMT QE shared tasks, are assumed to be symmetric, i.e., they penalise over and underestimates equally. This assumption is however too simplistic for many possible applications of QE. For example:

- In a *post-editing* scenario, a project manager may have translators with limited expertise in post-editing. In this case, automatic translations should not be provided to the translator unless they are highly likely to have very good quality. This can be enforced this by increasing the penalisation weight for underestimates of post-editing time. We call this the *pessimistic* scenario.

- In a *gisting* scenario, a company wants to automatically translate their product reviews so that they can be published in a foreign language without human intervention. The company would prefer to publish only the reviews translated well enough, but having more reviews published will increase the chances of selling products. In this case, having better recall is more important and thus only reviews with very poor translation quality should be discarded. We can accomplish this by heavier penalisation on overestimates, a scenario we call *optimistic*.

We can model these scenarios by employing *asymmetric* loss functions, which can penalise over and underestimates differently. They are parameterised by a weight w, which values defines if we addressing either a pessimistic or an optimistic setting. In this Section we show two asymmetric losses that are widely used in econometrics literature (Cain and Janssen, 1995; Zellner, 1986, *inter alia)*.

The asymmetric linear (henceforth, *alin*) loss is a generalisation of the absolute error:

$$L(\hat{y}, y) = \begin{cases} w(\hat{y} - y) & \text{if } \hat{y} > y \\ y - \hat{y} & \text{if } \hat{y} \leq y, \end{cases} \tag{5.3}$$

where $w > 0$ is the weight given to overestimates. If $w > 1$ we have a *pessimistic* scenario, where we favour underestimates. Conversely, for $0 < w < 1$ we obtain an *optimistic* scenario, which gives preference to overestimates. Finally, for $w = 1$ we retrieve the original symmetric absolute error loss.

Another asymmetric loss is the linear exponential or *linex* loss (Varian, 1975):

$$L(\hat{y}, y) = \exp[w(\hat{y} - y)] - (\hat{y} - y) - 1 \tag{5.4}$$

This loss attempts to keep a linear penalty in lower risk regions, while imposing an exponential penalty in the higher risk ones. Negative values for w will result in a pessimistic setting, while positive values will result in the optimistic one. For $w = 0$, the loss approximates a squared error loss. Usual values for w tend to be close to 1 or -1 since for higher weights the loss can quickly reach very large scores. Both alin and linex losses are shown on Figure 5.6.

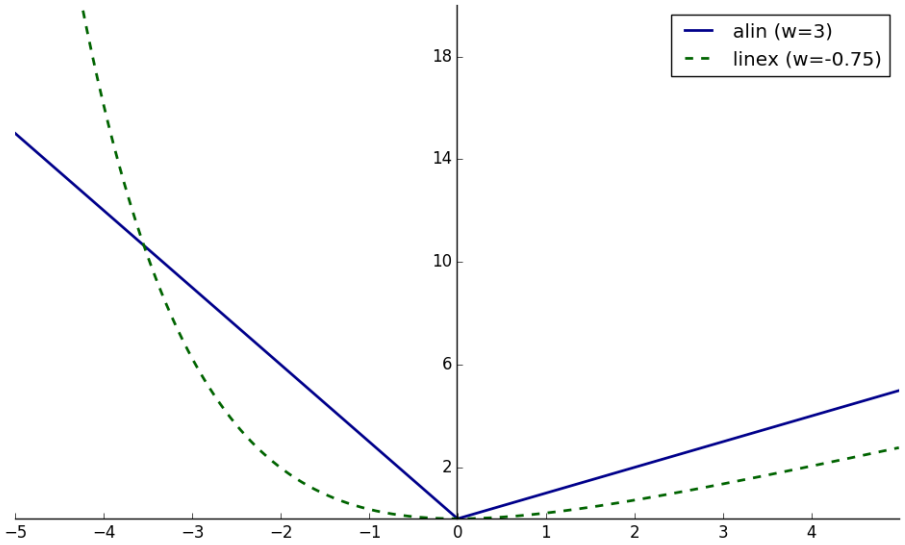


Fig. 5.6 Asymmetric losses. These curves correspond to the pessimistic scenario since they impose larger penalties when the prediction is lower than the true label. In the optimistic scenario the curves would be reflected with respect to the vertical axis.

In Section 2.4.1 we explained how predictive distributions can be converted into point estimates by the use of Bayes estimators for MAE and RMSE. The same reasoning can be applied for the asymmetric losses here. Here we use the derivations from Christoffersen and Diebold (1997), who derived Bayes estimators for both losses we introduced here. The best estimator for the alin loss is equivalent to the $\frac{w}{w+1}$ quantile of the predictive distribution. Note that we retrieve the median when $w = 1$, as expected. The best estimator for the linex loss can be easily derived and results in:

$$\hat{y} = \mu_y - \frac{w\sigma_y^2}{2}$$

where μ_y and σ_y^2 are the mean and the variance of the predictive posterior.

The use of Bayes estimators for asymmetric losses is a useful application for well-calibrated predictive distributions. An alternative would be to derive algorithms to directly optimise the loss. In the context of the alin loss, this is sometimes referred as *quantile*

regression (Koenker, 2005). However, such a setting would require full retraining in case the loss changes, while the Bayesian approach can reuse the same predictive posteriors through a different estimator. For instance, an expert might decide to change the value of w to fine tune the penalisation for over(under)estimates.

Experiments and analysis

Here we assess the models and datasets used in Section 5.3.3 in terms of their performance in the asymmetric setting. For this we use the same models used in the intrinsic evaluation experiments. We collect the predictions obtained using the 10-fold cross-validation protocol and apply different Bayes estimators corresponding to the asymmetric losses. Evaluation is performed using the same loss employed in the estimator (for instance, when using the linex estimator with $w = 0.75$ we report the results using the linex loss with same w) and averaged over the 10 folds.

To simulate both pessimistic and optimistic scenarios, we use $w \in \{3, 1/3\}$ for the alin loss and $w \in \{-0.75, 0.75\}$ for the linex loss. The only exception is the **en-de** dataset, where we report results for $w \in -0.25, 0.75$ for linex[2]. We also report results only for models using the Matèrn $5/2$ kernel. While we did experiment with different kernels and weighting schemes[3] our findings showed similar trends so we omit them for the sake of clarity.

Results for the optimistic scenario are given on Table 5.4. The $\tanh$-based warped GP models give consistently better results than standard GPs. The $\log$-based models also give good results for alin but for linex the results are mixed except for **en-es**. This is probably again related to the larger sizes of the fr-en and en-de datasets, which allows the $\tanh$-based models to learn richer representations.

The pessimistic scenario results, shown in Table 5.5, shows interesting trends. While the results for alin follow a similar pattern when compared to the optimistic setting, the results for linex are consistently worse than the standard GP baseline. A key difference between alin and linex is that the latter depends on the variance of the predictive distribution. Since the warped models tend to have less variance, we believe the estimator is not being "pushed" towards the positive tails as much as in the standard GPs. This makes the resulting predictions not conservative enough (i.e. the post-editing time predictions are lower) and this is heavily (exponentially) penalised by the loss. This might be a case where a standard GP

[2]Using $w = -0.75$ in this case resulted in loss values on the order of 10^7. In fact, as it will be discussed in the next Section, the results for the linex loss in the pessimistic scenario were inconclusive. However, we report results using a higher w in this case for completeness and to clarify the inconclusive trends we found.

[3]We also tried $w \in \{1/9, 1/7, 1/5, 5, 7, 9\}$ for the AL loss and $w \in \{-0.5, -0.25, 0.25, 0.5\}$ for the linex loss.

	English-Spanish		**French-English**		**English-German**	
	alin	linex	alin	linex	alin	linex
Standard GP	1.187	0.447	0.677	0.127	1.528	0.610
Warped GP						
log	1.060	0.299	0.675	0.161	1.457	0.537
tanh1	1.050	0.300	0.677	0.124	1.459	0.503
tanh2	1.054	0.300	0.671	0.121	1.455	0.504
tanh3	1.053	0.299	0.666	0.120	1.456	0.497

Table 5.4 Optimistic asymmetric scenario results. This setting corresponds to $w = 1/3$ for alin and $w = 0.75$ for linex.

	English-Spanish		**French-English**		**English-German**	
	alin	linex	alin	linex	alin	linex
Standard GP	1.633	3.009	0.901	0.337	2.120	0.217
Warped GP						
log	1.534	3.327	0.914	0.492	2.049	0.222
tanh1	1.528	3.251	0.901	0.341	2.064	0.220
tanh2	1.543	3.335	0.894	0.347	2.045	0.220
tanh3	1.538	3.322	0.886	0.349	2.042	0.219

Table 5.5 Pessimistic asymmetric scenario results. This setting uses $w = 3$ for alin and $w = -0.75$ for linex, except for English-German, where $w = -0.25$.

is preferred but can also indicate that this loss is biased towards models with high variance, even if it does that by assigning probability mass to nonsensical values (like negative time). We leave further investigation of this phenomenon for future work.

5.3.5 Active learning for Quality Estimation

As explained in Section 5.1, an ideal QE setting requires separate datasets and models for each possible translation scenario. Such a setting can be very expensive since it requires new annotation every time a domain-specific setting changes. In these cases, one way to reduce costs is by carefully selecting which data instances should be labelled.

Active Learning (AL) is the idea of letting the model choose the data from which it learns (Settles, 2012). The main hypothesis says that a model is able to perform better with less data if it is allowed to select instances according to some criterion. AL has been successfully

used in a number of NLP applications such as text classification (Lewis and Gale, 1994), named entity recognition (Vlachos, 2008) and parsing (Baldridge and Osborne, 2004).

In this Section we focus on *pool-based* AL, which assumes there is a pool of unlabelled data and the model can *query* all or some of instances in that pool. This querying can be done using many different criteria but the most common one is *uncertainty sampling* (Lewis and Gale, 1994), which queries a label for the instance which the model is most uncertain about. In regression, the variance of the predictive distribution is usually a good measure of uncertainty. In fact, in the case of Gaussians, selecting the instance with the highest variance is equivalent to selecting the one with the highest entropy with respect to the predictive distribution (Settles, 2012). While many methods for query selection do exist in the literature (Olsson, 2009; Settles, 2012), here we focus on uncertainty sampling only. Nevertheless, the information provided by our models can be used in other query settings.

To assess how our models fare in AL settings, we first compare the GP models introduced in Section 5.3.2 between themselves, except that we use isotropic kernels instead. The reason for this is to reduce the number of hyperparameters, which is paramount since this setting deals with very small dataset sizes. Our main hypothesis is that the AL procedure will generate a more accurate model with less instances. In this sense, our evaluation is based on performance curves, where we show the variation in a number of metrics as the training set size increases.

Experiments and analysis

Our experiments are simulations, in the sense that we already have labels for all instances in the pool. In each dataset we use a random split of $90\%/10\%$ for training and test, and initialise each model with a random subset of 50 instances from the training set. This model then predicts response variable distributions for all sentence pairs in the pool, which is composed of the remaining training set instances. At each iteration, the sentence pair which gets the highest predictive variance is added to training set, the model is retrained and we calculate NLPD, MAE and Pearson's r on the test set. This will result in what we call performance curves and we expect that AL will reach better performance with smaller training set sizes, compared to a model that increments the training set with randomly selected instances, a common baseline in the AL literature (Settles, 2012).

Our first finding is that the Warped GP models trained with $\tanh 2$ and $\tanh 3$ gave very noisy and unstable curves and on average performed worse than the other models. This is likely to be due to their increased number of hyperparameters, which tends to make optimisation harder with the small datasets we use in this setting.

For the remaining models we show the performance curves on Figure 5.7. As expected, NLPD for the Standard GPs is higher in all pairs, since they overestimate the support of the distributions. Other metrics show mixed trends, depending on the language pair. For **en-es**, the Warped GP with $\log$ function consistently outperforms the other models for point estimate metrics. This trend is reversed for **fr-en** though, where the Standard GP and the Warped GP with $\tanh 1$ perform the best, with no clear difference between them. Notice also the unstable behaviour of $\tanh 1$ for small datasets, due to difficulties arisen in the optimisation. Finally, for **en-de** we obtain an interesting trend: while $\log$ gives the best NLPD, $\tanh 1$ results in better point estimate metrics after reaching 200 instances.

For our second set of experiments, we take the best performing Warped GP model in the previous experiment and compare it with a model that selects random instances. For both AL and random curves the model is the same, only the query method (and therefore the training data) differs. The results, shown in Figure 5.8, show again mixed trends. The AL method clearly outperforms random selection for **en-es** but the opposite happens for **fr-en**, except for NLPD. For **en-de**, random selection fares better in small datasets but AL catches up and outperforms it when reaching around 350 instances.

Overall, these results show that the best AL setting, including which underlying model to use, is very dependent on the language pair. Clearly, 50 instances is enough to get accurate uncertainty estimates for AL in **en-es** but not for **en-de**, which requires around 350 instances. Finally, for **fr-en** the results were mostly inconclusive as we did not see any benefits from AL. For real world scenarios, these findings motivate preliminary, simulated AL benchmarks for a new QE setting before applying it in practical tasks.

5.4 Tree Kernels for Quality Estimation

The models investigated in the previous Section use a set of hand-crafted features. In this Section our goal is to move away from a feature engineering perspective by using kernels directly defined on the sentence pairs. We focus on representing each sentence through its constituency tree, as syntax can be an useful indicator of quality. This allow us to employ models based on the tree kernels we introduced in Section 3.3.

Tree kernels were used before in QE by Hardmeier (2011) and Hardmeier et al. (2012), with promising results. His best performing model combined tree kernels with a set of hand-crafted features but good results were found by employing tree kernels only. However, his method used SVMs as the underlying algorithm and hyperparameters were fixed. Here

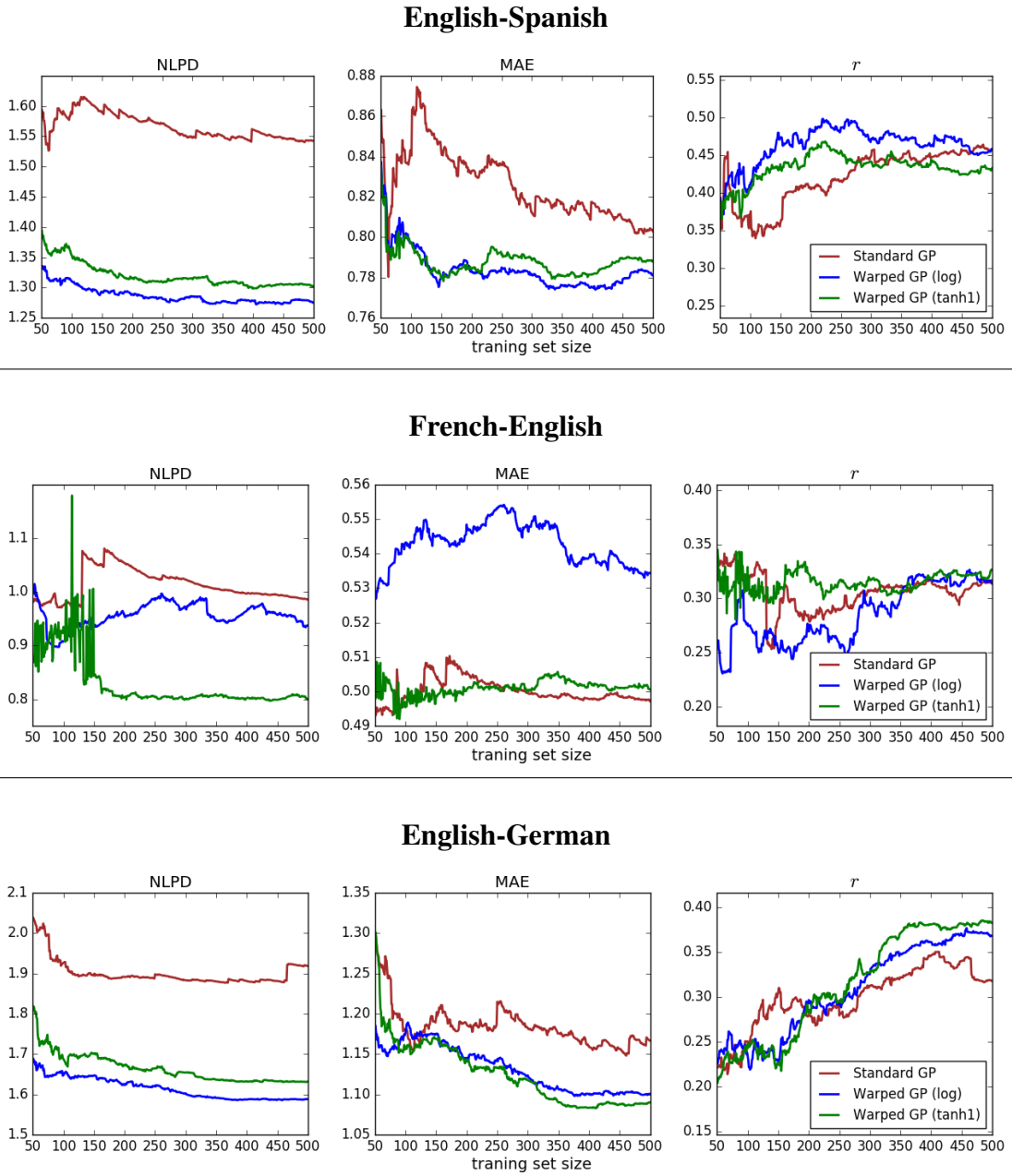


Fig. 5.7 Performance curves for the AL experiments using different GP models. For each language pair we show how a specific metric changes as instances are added to training set. The left column shows NLPD, the middle one corresponds to MAE and the right column shows Pearson's r scores.

we propose to use GPs and employ the extensions we introduced in Section 3.3 to learn tree kernel hyperparameters, aiming at capturing better syntactic representations.

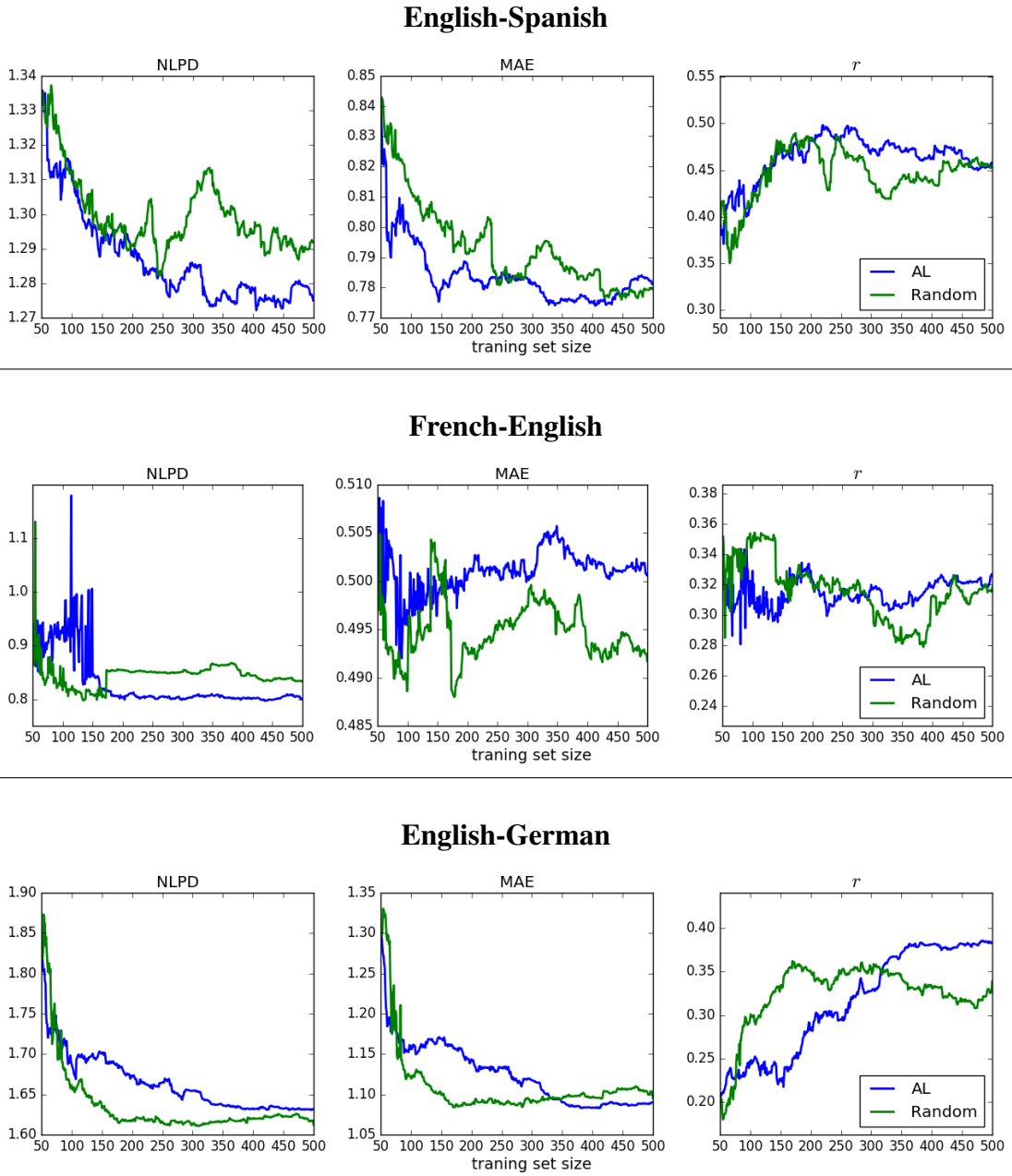


Fig. 5.8 Performance curves for the AL vs. random instance selection experiments. All models are Warped GPs, with the $\log$ function for English-Spanish and $\tanh 1$ for French-English and English-German.

5.4.1 Experimental settings

We benchmark our approach using the **en-es** and **fr-en** datasets, using a random $90\%/10\%$ split for training/test and evaluating using NLPD, MAE and Pearson's r. The models are composed of Warped GPs with a $\log$ warping function with different tree kernel combinations.

The constituency trees for all language pairs are obtained using the Stanford parser (Manning et al., 2014).

Two sets of experiments are performed, one using a combination of SSTKs and another using a combination of SASSTKs, the symbol-aware version introduced in Section 3.3.1. All tree kernels are normalised, following the procedure shown in Section 3.3.3.

Since our data is composed of pairs of trees we use a linear combination of two tree kernels:

$$k(\mathbf{x}, \mathbf{x}') = (b_{src} \times k(t_{src}, t'_{src})) + \left(b_{tgt} \times k(t_{tgt}, t'_{tgt})\right), \tag{5.5}$$

where $k(t, t')$ correspond to either source or target tree kernels and b are bias terms. The role of the biases is not only to balance contributions between kernels but also to scale the output of the whole kernel. This is essential in order to achieve good results as we use normalised tree kernels, which return values in the $[0, 1]$ range.

For the SASSTK models we can define symbol-specific hyperparameters. Instead of having one for each symbol presented in the constituency trees we follow one of the ideas discussed in Section 3.3.2, where we select the three most frequent symbols for each language (excluding pre-terminals). Then we assign symbol-specific hyperparameters for each of these symbols and use default ones for the others.

The most frequent symbols differ depending on the dataset. For **en-es**, we have NP, VP and S for English and $\mathrm{grup.nom}, \mathrm{sn}$ and spec for Spanish. For **fr-en**, NP, PP and VN are the most frequent symbols for French and NP, VP and PP the ones for English. Finally, for **en-de** we have NP, VP and S for English and NP, S and PP for German. In summary, most of these symbols refer to noun and verb phrases and we expect that their high frequency will help guide the tree kernels towards a setting which can extract more signal from these trees.

Our baselines are feature-based models using the 17 QuEst features. We compare with models trained using linear and Matèrn $5/2$ kernels.

5.4.2 Results and analysis

In Table 5.6 we report the results of our benchmarks. The main finding is that the models based on tree kernels were outperformed by the feature-based models in all language pairs. This is in contrast with the findings of Hardmeier et al. (2012), which show similar performance between tree kernels and feature-based models. An explanation for this is the different nature of the labels: Hardmeier et al. (2012) focus on predicting Likert scores in the $[1 - 5]$ range. Unlike post-editing rates, these scores are more coarse-grained and capture different aspects of the evaluation. Overall, the results show evidence that constituency trees alone

	English-Spanish			French-English			English-German		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Linear	1.480	0.937	0.421	0.935	0.540	0.293	1.847	1.349	0.412
Matèrn $5/2$	1.275	0.784	0.441	0.805	0.489	0.397	1.558	1.069	0.438
SSTK	1.346	0.824	0.324	0.838	0.494	0.265	1.595	1.093	0.316
SASSTK	1.340	0.836	0.313	0.851	0.503	0.226	1.611	1.107	0.285

Table 5.6 Results for the tree kernel experiments. All models are Warped GPs with $\log$ as warping function and using a combination of either two SSTKs or two SASSTKs. The Matèrn $5/2$ baseline uses the 17 QuEst features.

lose essential information in predicting post-editing rates compared to the feature-based models.

One of our hypothesis was that the SASSTK would give better results than the SSTK, since it inherently has larger model capacity. However, surprisingly that was not the case. Inspecting the marginal likelihoods we noted that the SASSTK models are able to achieve higher values compared to the SSTK, for all language pairs. For instance, in *en-es*, the marginal likelihood for the SSTK was -1031.83, while it reached -1026.59 for SASSTK. This shows that the SASSTK lower performance is not an optimisation problem, but is evidence of *overfitting*.

A possible explanation for this overfitting behaviour is that the constituency trees alone are not discriminative for the task in general. In this case, a fine-grained version such as the SASSTK will not enhance the performance. But since it has increased model capacity, the empirical Bayes procedure will eventually find a better value for the marginal likelihood, which in turn will not translate into better performance in unseen data because the underlying features are actually mostly noise.

5.4.3 Combining trees and hand-crafted features

Given that we have evidence that constituency trees alone are not enough to reach good predictive performance, a natural follow-up is to investigate if they can be plugged into models that already work. In our case, we can do that by simply extended the linear combination showed in Equation 5.5 by

$$k(\mathbf{x}, \mathbf{x}') = (b_{src} \times k(t_{src}, t'_{src})) + \big(b_{tgt} \times k(t_{tgt}, t'_{tgt})\big) + \big(b_{feat} \times k_{feat}(\mathbf{x}_{feat}, \mathbf{x}'_{feat})\big)\,, \quad (5.6)$$

where k_{feat} is a feature-based kernel on vectors $\mathbf{x}_{feat}$. A similar combination obtained the best results in Hardmeier et al. (2012).

	English-Spanish			French-English			English-German		
	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑	NLPD ↓	MAE ↓	r ↑
Linear	1.480	0.937	0.421	0.935	0.540	0.293	1.847	1.349	0.412
Matèrn $5/2$	1.275	0.784	0.441	0.805	0.489	0.397	1.558	1.069	0.438
SSTK + Matèrn $5/2$	1.256	0.777	0.450	0.787	0.479	0.387	1.529	1.030	0.472
SASSTK + Matèrn $5/2$	1.280	0.824	0.362	0.802	0.482	0.374	1.530	1.030	0.459

Table 5.7 Results for the experiments combining tree kernels with feature-based kernels. All models are Warped GPs with $\log$ as warping function and using a combination of either two SSTKs or two SASSTKs and a Matèrn $5/2$ kernel on the 17 QuEst features.

Here we benchmark this approach using both SSTK and SASSTK, where the feature-based kernel is a Matèrn $5/2$. In these experiments, we start from the optimised model with the features only, fix their corresponding lengthscale hyperparameters, add the tree kernels to the model and retrain it. That way, we can ensure we are starting from a model we already know it works, but still allow the respective biases terms to vary, reflecting individual contributions from each kernel.

Table 5.7 shows the results. In general, the combination of SSTK and the Matèrn $5/2$ outperformed the baseline for all language pairs. This is an encouraging result, since it shows that while constituency trees on their own are not very discriminative, they can contribute to feature-based models. The results using the SASSTK showed again evidence of overfitting, as we obtained similar trends with the marginal likelihood values as in Section 5.4.2.

5.4.4 Inspecting hyperparameters

To obtain more insights about the tree kernel models we inspected the obtained hyperparameter values for the combination of an SSTK and Matèrn $5/2$, in all three language pairs. The values shown on Table 5.8 point out interesting trends. First, analysing the bias terms, as expected the feature-based kernels have larger weights, corresponding to their higher discriminative power. Comparing the tree kernels only, we can see that the corresponding hyperparameter values differ depending on the dataset. For *fr-en* and *en-de*, the target tree kernel has higher weight, which can be interpreted as the model relying more on target sentence fluency than on source sentence complexity. The opposite trend is shown for *en-es*, which might be explained by the smaller dataset size compared to the other language pairs.

The internal tree kernel hyperparameters give further information on the model. For the source tree, the λ values are close to zero, which means that it heavily penalises larger tree fragments. The α hyperparameters show that for **fr-en** it prefers a representation close

	English-Spanish	**French-English**	**English-German**
b_{feat}	0.658	15.208	0.451
b_{src}	0.037	0.146	0.051
b_{tgt}	0.008	0.225	0.086
λ_{src}	9.9×10^{-6}	2.5×10^{-6}	1.8×10^{-6}
α_{src}	0.504	1.041	0.625
λ_{tgt}	0.005	0.568	0.044
α_{tgt}	0.035	0.411	3.442

Table 5.8 Optimised hyperparameter values for the SSTK + Matèrn $5/2$ combination.

to the original SSTK, but for the other pairs it decides on lower values, which gives more preference to full subtrees.

In the target side we see different trends. The λ values are in general higher, showing a preference for larger fragments. This is specially the case for **fr-en**, which ended up with λ higher than one. The α hyperparameters vary significantly across language pairs. The low value in the case of **en-es** emphasizes full subtrees while we get the opposite behaviour for **en-de**, which makes use of more varied kinds of fragments.

The main finding we get from inspecting these models is that optimal learned representations can vary greatly across language pairs. In this sense, the main benefit of our approach is to be able to balance between these different representations in a sensible manner.

5.5 Discussion

In this Chapter we introduced Gaussian Process regression models for Machine Translation Quality Estimation. We focused on post-editing rate prediction as a quality metric as it has direct connections to productivity in translation environments. Our goals consisted in obtaining better uncertainty estimation for quality metrics and move away from hand-crafted feature-based models by representing sentences as their constituency trees and employing models based on tree kernels.

The first goal was successfully achieved by the use of Warped GPs. Intrinsic evaluation showed these models were able to better estimate the distribution of response variables, including their support. We further showed how the probabilistic predictions obtained from Warped GPs could be applied in two real world applications, one that addressed asymmetric risk scenarios and one that aimed at building models with scarce data using active learning.

For the former, we used asymmetric loss functions and demonstrated how they can be applied to predictive distributions by the use of Bayes estimators. Results showed that in general more accurate distributions result in better performance with respect to these asymmetric losses. For the active learning scenario our simulations showed that some language pairs can benefit from the better uncertainty estimates provided by Warped GPs.

For the second goal, the tree kernel approach did not outperform the feature-based models, showing that we are still not in a position of getting rid of hand-crafted features. However, we also showed that we can obtain the best results by combining the two approaches, which is a very promising finding. A next step would be to understand what kind of feature information is not captured by the constituency trees and investigate how it can be encoded in other kind of structural kernels. We also showed how the tree kernel models can be inspected in order to obtain insights on what kind of representations they are learning. This gave us evidence that representations can differ greatly depending on the language pair.

Finally, we also obtained negative results with respect to the symbol-aware approach for tree kernels. Specifically, our analysis show evidence of overfitting. A possible avenue for investigation is to move away from empirical Bayes approaches when dealing with such kernels. Alternatives include the use of maximum-a-posteriori estimation through the use of hyperpriors or integrate hyperparameters out (Murray and Adams, 2010; Osborne, 2010). This can prove challenging though, considering the computational cost of these kernels.

Chapter 6

Conclusion

This thesis investigated the problem of Text Regression, which aims at predicting indicators from linguistic inputs in the form of continuous response variables. Our goals focused on two aspects which were mostly understudied in previous work: dealing with the inherent uncertainty present in the response variables and enhancing input text representations. To achieve these goals we proposed the use of Gaussian Processes, a probabilistic kernelised framework. The Bayesian aspect of GPs allowed us to incorporate uncertainty through well-calibrated predictive distributions, while the kernelised component affords elegant means of processing text inputs by incorporating structural kernels. We evaluated our approach in two real world applications, Emotion Analysis and Machine Translation Quality Estimation, which confirmed the validity and usefulness of our approach.

6.1 Summary

We now revisit the contributions we stated in Section 1.3.

Text Regression models that provide well-calibrated predictive distributions.

Introduced in Chapter 2, Gaussian Processes have been widely used in regression problems. In this thesis we provided the first comprehensive evaluation on its applicability for textual inputs. GPs allow predictions to be represented as well-calibrated distributions that capture uncertainty in the response variables, a property that is not available in other widely used regression models such as SVMs. Specifically, we capitalise on their flexibility to propose a range of GP-based models and evaluate them in terms of their Negative Log Predictive Density. Our findings show that choices of kernels and other enhancements such as Warped GPs can

substantially affect the predictive distributions, as evidenced by the improvements obtained in the benchmark tasks.

A novel kernel framework for textual data. In Chapter 3 we proposed to combine GPs with kernels designed to capture similarity between discrete structures, such as strings or trees. Our approach uses the idea of optimising GP hyperparameters through empirical Bayes, which is based on maximising the model marginal likelihood. We derived the gradients of string and tree kernels, allowed this elegant model selection procedure to be applied for these kernels as well, a novel contribution of this thesis. This approach not only allowed us to fine tune kernel hyperparameters but also provided the means to enhance the capacity of these kernels through more fine-grained parameterisations. Empirical evidence through a series of synthetic data experiments validated our approach, showing it can reliably learn model hyperparameters with a reasonable amount of data, in the order of hundreds of instances. Furthermore, in the case of strings, we also proposed a new algorithm based on vectorisation, which capitalises on recent advances in linear algebra library implementations and parallelisation through multi-core devices such as GPUs.

New methods for Emotion Analysis. Our first real world benchmark was Emotion Analysis, where we proposed two contributions in Chapter 4. The first one used a combination of GPs and string kernels, using inputs represented as sequences of word vectors. Our findings showed promising results for this approach, outperforming linear models based on averaging embeddings as features. This showed that the string kernels were able to capture information that is lost in simple bag-of-words averaging. We also compared with non-linear GP models applied to averaged word vectors, which obtained better performance compared to our approach. This is evidence that the linear nature of the string kernels is too restrictive and there is benefit in pursuing further directions in making these kernels non-linear.

The second contribution was a joint emotion model using multi-task GPs. Unlike the single-task models which are trained on a per-emotion basis, the joint model can take into account correlations and anti-correlations between emotional labels. In terms of predictive performance we did not see significant differences between our approach and the single-task models. Nevertheless, the joint model has the additional benefit of learning emotion covariances which can be inspected for interpretability.

Novel approaches for Machine Translation Quality Estimation. The second evaluation setting was Machine Translation Quality Estimation, where two contributions were made in Chapter 5. First, we proposed the use of Warped GPs with the goal of obtaining improved uncertainty estimates over post-editing rate predictions. Our findings show major improvements in performance compared to standard GPs, with Warped GPs providing state-of-the-art predictive distributions. In order to provide an extrinsic evaluation and also motivate the need for uncertainty estimates, we also showed two applications where this information is useful, one that address scenarios with asymmetric risk and another that employs active learning for reducing annotation efforts.

Our second contribution was a model based on combining GPs with tree kernels, in order to model syntactic aspects of this task. Specifically, we benchmark a set of different tree kernels, including a novel symbol-aware approach that fine-tune hyperparameters with respect to syntactic constituents. Our findings show that tree kernels on their own are outperformed by feature-based models but we obtained consistent improvements when combining the two approaches. Furthermore we also showed that the symbol-aware approach lends itself to model interpretability, as it can be probed to give insights on what constituents are more discriminative for the task.

6.2 Future Research

The work presented here also provide the grounds for many directions of future research. We mention some of these directions here:

Structural kernel extensions Many other variants of the string and tree kernels exist in the literature and the GP model selection approach can also be applied to these variants, as long as their gradients are available. For tree kernels, the Partial Tree Kernel (Moschitti, 2006a) is a natural extension which could be combined with GPs. Soft matching approaches have also been proposed in previous work (Bloehdorn and Moschitti, 2007; Croce et al., 2011). For string kernels, Cancedda et al. (2003) considered the idea of having symbol specific decay hyperparameters, much in line with our Symbol-aware Subset Tree Kernel. None of these previous works focused on learning hyperparameters with respect to the task, employing either standard values or using lexical frequency information. Main challenges for this avenue include finding ways

to mitigate overfitting due to the larger number of hyperparameters and to ensure optimisation procedures do not lead to spurious local optima.

Non-linear structural kernels A key finding in Chapter 4 was the better performance of non-linear kernels compared to our string kernel approach. This shows evidence that in real world tasks some phenomena can not be captured by linear patterns and motivate the need for non-linearities also in structural kernels. A promising approach to reach this is based on Arc-cosine kernels (Cho and Saul, 2009), which are inspired by neural network architectures. It has been proven by Neal (1996) that a single-layer neural network with Gaussian distributed weights and sigmoid non-linearities becomes equivalent to a GP when the number of hidden units is assumed to be infinite. Arc-cosine kernels builds up on this idea to propose models equivalent to multi-layer neural networks with different non-linearities. The core calculation of this kernel is a dot-product, which could be replaced by any kind of structural kernel we presented in this thesis.

Improved models The literature on GPs is vast and many other extensions exist that could also be explored in Text Regression. For better treatment of uncertainty, one direction is to move away from an empirical Bayes approach and integrate over hyperparameters. In the case of kernel hyperparameters, this can be done via sampling using MCMC methods (Murray and Adams, 2010). For Warped GPs, a Bayesian extension was proposed by Lázaro-Gredilla (2012), which employs variational methods for inference. Another direction is to go beyond the ICM model for multi-task learning, to allow more complex combinations (Álvarez et al., 2012).

Scalability The focus of this thesis was on problems where data availability is scarce. A natural question is if our approach can be used in big data scenarios. Standard GPs do not scale to these settings due to the Gram matrix inversion bottleneck, cubic in the number of instances. There is large body of work which aims at scaling GPs to big data: a comprehensive survey on this is done by Quiñonero-Candela et al. (2007). Many methods are based on low-rank approximations to the Gram matrix and using the so-called *inducing points*, which are pseudo-instances optimised using a variational objective (Hensman et al., 2013; Titsias, 2009). However this approach assumes that inputs lie in a continuous space: extending this to the discrete structural world can be a daunting challenge but which can enable the use of GPs with structural kernels in big data scenarios.

Task-specific extensions Finally, some other avenues could be explored in the benchmark tasks we addressed in our thesis. For Emotion Analysis, a natural extension is to employ emotion lexicons instead of, or in conjunction with distributional word embeddings. Extending our approach to song lyrics is also an interesting direction as the GP models can be enhanced to take into account the sequential nature of this data. For Quality Estimation, an exciting avenue is to move away from feature-based models: the tree kernel approach we proposed here is a step in that direction. While syntactic trees can capture fluency and complexity aspects, ideally we should also have a component that takes translation *adequacy* into account. In terms of kernels, a possible direction is to adapt pairwise string kernels, which were previously developed for modelling interactions between proteins in bioinformatics (Ben-Hur and Noble, 2005; Kashima et al., 2009). These can be combined with bilingual word vectors (Gouws et al., 2015; Vulić and Moens, 2015) and soft matching to model if a translation can keep the meaning of the corresponding original sentence.

6.3 Final Remarks

In summary, this thesis showed the feasibility of using Gaussian Processes for modelling textual data in a regression setting. We showed this approach can bring benefits in uncertainty estimation and allow flexible text representations. Many interesting avenues for future work exist: our hope is that this thesis will provide the theoretical and empirical grounds for such directions.

References

Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. (2014). A Reliable Effective Terascale Linear Learning System. *Journal of Machine Learning Research*, 15:1111–1133.

Alm, C. O., Roth, D., and Sproat, R. (2005). Emotions from text: machine learning for text-based emotion prediction. In *Proceedings of EMNLP*, pages 579–586.

Altun, Y., Hofmann, T., and Smola, A. J. (2004). Gaussian Process Classification for Segmenting and Annotating Sequences. In *Proceedings of ICML*, page 8. ACM Press.

Álvarez, M. A., Rosasco, L., and Lawrence, N. D. (2012). Kernels for Vector-Valued Functions: a Review. *Foundations and Trends in Machine Learning*, pages 1–37.

Bach, N., Huang, F., and Al-Onaizan, Y. (2011). Goodness: A Method for Measuring Machine Translation Confidence. In *Proceedings of ACL*, pages 211–219.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation By Jointly Learning To Align and Translate. In *Proceedings of ICLR*, pages 1–15.

Baldridge, J. and Osborne, M. (2004). Active learning and the total cost of annotation. In *Proceedings of EMNLP*, pages 9–16.

Beck, D. (2014). Bayesian Kernel Methods for Natural Language Processing. In *Proceedings of ACL Student Research Workshop*, pages 1–9.

Beck, D., Cohn, T., Hardmeier, C., and Specia, L. (2015). Learning Structural Kernels for Natural Language Processing. *Transactions of the Association for Computational Linguistics*, 3:461–473.

Beck, D., Cohn, T., and Specia, L. (2014a). Joint Emotion Analysis via Multi-task Gaussian Processes. In *Proceedings of EMNLP*, pages 1798–1803.

Beck, D., Shah, K., Cohn, T., and Specia, L. (2013a). SHEF-Lite : When Less is More for Translation Quality Estimation. In *Proceedings of WMT13*, pages 337–342.

Beck, D., Shah, K., and Specia, L. (2014b). SHEF-Lite 2.0 : Sparse Multi-task Gaussian Processes for Translation Quality Estimation. In *Proceedings of WMT14*, pages 307–312.

Beck, D., Specia, L., and Cohn, T. (2013b). Reducing Annotation Effort for Quality Estimation via Active Learning. In *Proceedings of ACL*.

Beck, D., Specia, L., and Cohn, T. (2016). Exploring Prediction Uncertainty in Machine Translation Quality Estimation. In *Proceedings of CoNLL*.

Ben-Hur, A. and Noble, W. S. (2005). Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21(Suppl 1):i38–i46.

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4.

Blatz, J., Fitzgerald, E., Foster, G., Gandrabur, S., Goutte, C., Kulesza, A., Sanchis, A., and Ueffing, N. (2004). Confidence estimation for machine translation. In *Proceedings of the 20th Conference on Computational Linguistics*, pages 315–321.

Bloehdorn, S. and Moschitti, A. (2007). Exploiting Structure and Semantics for Expressive Text Kernels. In *Proceedings of CIKM*.

Bojar, O., Buck, C., Callison-Burch, C., Federmann, C., Haddow, B., Koehn, P., Monz, C., Post, M., Soricut, R., and Specia, L. (2013). Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of WMT13*, pages 1–44.

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-amand, H., Soricut, R., Specia, L., and Tamchyna, A. (2014). Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of WMT14*, pages 12–58.

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Jimeno Yepes, A., Koehn, P., Logacheva, V., Monz, C., Negri, M., Neveol, A., Neves, M., Popel, M.,

Post, M., Rubino, R., Scarton, C., Specia, L., Turchi, M., Verspoor, K., and Zampieri, M. (2016). Findings of the 2016 Conference on Machine Translation. In *Proceedings of WMT*, volume 2, pages 131–198.

Bonilla, E. V., Chai, K. M. A., and Williams, C. K. I. (2008). Multi-task Gaussian Process Prediction. *Advances in Neural Information Processing Systems*.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classiiers. In *Proceedings of COLT*, pages 144–152.

Bratières, S., Quadrianto, N., and Ghahramani, Z. (2013). Bayesian Structured Prediction using Gaussian Processes. *arXiv:1307.3846*, pages 1–17.

Brown, P., Cocke, J., Pietra, S., and Pietra, V. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Cain, M. and Janssen, C. (1995). Real Estate Price Prediction under Asymmetric Loss. *Annals of the Institute of Statististical Mathematics*, 47(3):401–414.

Callison-Burch, C., Koehn, P., Monz, C., Post, M., Soricut, R., and Specia, L. (2012). Findings of the 2012 Workshop on Statistical Machine Translation. In *Proceedings of WMT12*.

Cancedda, N., Gaussier, E., Goutte, C., and Renders, J.-M. (2003). Word-Sequence Kernels. *The Journal of Machine Learning Research*, 3:1059–1082.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, 28:41–75.

Cho, Y. and Saul, L. K. (2009). Kernel Methods for Deep Learning. In *Proceedings of NIPS*, pages 1–9.

Christoffersen, P. F. and Diebold, F. X. (1997). Optimal Prediction Under Asymmetric Loss. *Econometric Theory*, 13(06):808–817.

Clore, G. L., Ortony, A., and Foss, M. A. (1987). The psychological foundations of the affective lexicon. *Journal of Personality and Social Psychology*, 53(4):751–766.

Cohn, T. and Specia, L. (2013). Modelling Annotator Bias with Multi-task Gaussian Processes: An Application to Machine Translation Quality Estimation. In *Proceedings of ACL*, pages 32–42.

Collins, M. (2002). Discrimative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP*, pages 1–8.

Collins, M. and Duffy, N. (2001). Convolution Kernels for Natural Language. In *Proceedings of NIPS*, pages 625–632.

Collins, M. and Duffy, N. (2002). New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL*, number July, pages 263–270.

Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3):273–297.

Croce, D., Moschitti, A., and Basili, R. (2011). Structured Lexical Similarity via Convolution Kernels on Dependency Trees. In *Proc. of EMNLP*.

Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of ACL*.

de Souza, J. G. C., Negri, M., Ricci, E., and Turchi, M. (2015). Online Multitask Learning for Machine Translation Quality Estimation. In *Proceedings of ACL*, number 1, pages 219–228.

de Souza, J. G. C., Turchi, M., and Negri, M. (2014). Machine Translation Quality Estimation Across Domains. In *Proceedings of COLING*, pages 409–420.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society For Information Science*, 41.

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, pages 128–132.

Ekman, P. (1993). Facial Expression and Emotion. *American Psychologist*, 48(4):384–392.

Finkel, J. R., Kleeman, A., and Manning, C. D. (2008). Feature-based, Conditional Random Field Parsing. In *Proceedings of ACL*, pages 959–967.

Gouws, S., Bengio, Y., and Corrado, G. (2015). BilBOWA: Fast Bilingual Distributed Representations without Word Alignments. In *Proceedings of ICML*.

Graham, Y. (2015). Improving Evaluation of Machine Translation Quality Estimation. In *Proceedings of ACL*.

Hardmeier, C. (2011). Improving Machine Translation Quality Prediction with Syntactic Tree Kernels. In *Proceedings of EAMT*, pages 233–240.

Hardmeier, C., Nivre, J., and Tiedemann, J. (2012). Tree Kernels for Machine Translation Quality Estimation. In *Proceedings of WMT12*, pages 109–113.

Harris, Z. S. (1954). Distributional Structure. *Word*, 10(2-3):146–162.

Haussler, D. (1999). Convolution Kernels on Discrete Structures. Technical report, University of California at Santa Cruz.

He, Y., Ma, Y., van Genabith, J., and Way, A. (2010). Bridging SMT and TM with Translation Recommendation. In *Proceedings of ACL*, pages 622–630.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian Processes for Big Data. In *Proceedings of UAI*, pages 282–290.

Igel, C., Glasmachers, T., Mersch, B., and Pfeifer, N. (2007). Gradient-based Optimization of Kernel-Target Alignment for Sequence Kernels Applied to Bacterial Gene Start Detection. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):216–226.

Jaakkola, T., Diekhans, M., and Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7:95–114.

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of ECML*, volume 1398, pages 137–142.

Joshi, M., Das, D., Gimpel, K., and Smith, N. A. (2010). Movie Reviews and Revenues: An Experiment in Text Regression. In *Proceedings of NAACL*.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *arXiv*.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing*.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent Continuous Translation Models. In *Proceedings of EMNLP*, pages 1700–1709.

Kashima, H., Oyama, S., Yamanishi, Y., and Tsuda, K. (2009). On pairwise kernels: An efficient alternative and generalization analysis. In *Proceedings of PAKDD*, volume 5476 LNAI, pages 1030–1037.

Keltner, D. and Ekman, P. (2003). Facial Expression of Emotion. *Handbook of Affective Sciences*, pages 415–432.

Kim, Y. E., Schmidt, E. M., Migneco, R., Morton, B. G., Richardson, P., Scott, J., Speck, J. A., and Turnbull, D. (2010). Music Emotion Recognition : a State of the Art Review. In *Proceedings of ISMIR*, pages 255–266.

Koehn, P. (2005). Europarl : A Parallel Corpus for Statistical Machine Translation. In *Proceedings of MT Summit*, volume 11, pages 79—-86.

Koehn, P. (2010). *Statistical Machine Translation*. Cambridge University Press, 1st edition.

Koenker, R. (2005). *Quantile Regression*. Cambridge University Press.

Koponen, M., Aziz, W., Ramos, L., and Specia, L. (2012). Post-editing time as a measure of cognitive effort. In *Proceedings of WPTP*.

Lafferty, J., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of ICML*, pages 282–289.

Lázaro-Gredilla, M. (2012). Bayesian Warped Gaussian Processes. In *Proceedings of NIPS*, pages 1–9.

Leslie, C., Eskin, E., Weston, J., and Noble, W. S. (2003). Mismatch String Kernels for SVM Protein Classification. In *Proceedings of NIPS*.

Levy, O. and Goldberg, Y. (2014). Dependency-based Word Embeddings. In *Proceedings of ACL*, volume 2, pages 302–308.

Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1–10.

Lewis, D. D., Lewis, D. D., Schapire, R. E., Schapire, R. E., Callan, J. P., Callan, J. P., Papka, R., and Papka, R. (1996). Training Algorithms for Linear Text Classifiers. In *Proceedings of SIGIR*, pages 298–306.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text Classification using String Kernels. *The Journal of Machine Learning Research*, 2:419–444.

Lukasik, M., Cohn, T., and Bontcheva, K. (2015). Point Process Modelling of Rumour Dynamics in Social Media. In *Proceedings of ACL*, pages 518–523.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of EMNLP*, page 11.

Mahé, P. and Cancedda, N. (2008). Factored sequence kernels. In *Proceedings of ESANN*, number April, pages 23–25.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of ACL Demo Session*, pages 55–60.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Mihalcea, R. and Strapparava, C. (2012). Lyrics, Music, and Emotions. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 590–599.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of NIPS*, pages 1–9.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

Minka, T. P. (2001). *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology.

Mohammad, S. M. and Turney, P. D. (2011). Crowdsourcing a Word – Emotion Association Lexicon. *Computational Intelligence*, 59(000):1–24.

Moreau, E. and Vogel, C. (2014). Limitations of MT Quality Estimation Supervised Systems: The Tails Prediction Problem. In *Proceedings of COLING*, pages 2205–2216.

Moschitti, A. (2006a). Efficient Convolution Kernels for Dependency and Constituent Syntactic Trees. In *Proceedings of ECML*, pages 318–329.

Moschitti, A. (2006b). Making Tree Kernels practical for Natural Language Learning. In *Proceedings of EACL*, pages 113–120.

Murphy, K. P. (2012). *Machine Learning: a Probabilistic Perspective*.

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. In *Proceedings of NIPS*.

Murray, I. R. and Arnott, J. L. (1993). Toward the simulation of emotion in synthetic speech: A review of the literature on human vocal emotion. *The Journal of the Acoustical Society of America*, 93(2):1097.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer.

Oatley, K. (1989). The importance of being emotional. *New Scientist*, 123(1678).

Olsson, F. (2009). A literature survey of active machine learning in the context of natural language processing. Technical report.

Ortony, A., Clore, G. L., and Foss, M. A. (1987). The Referential Struture of the Affective Lexicon. *Cognitive Science*, 11:341–364.

Osborne, M. (2010). *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, University of Oxford.

Pang, B. and Lee, L. (2008). Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Duborg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*, pages 1532–1543.

Picard, R. W. (1997). *Affective Computing*. MIT Press.

Plank, B. and Moschitti, A. (2013). Embedding Semantic Similarity in Tree Kernels for Domain Adaptation of Relation Extraction. In *Proceedings of ACL*, pages 1498–1507.

Plitt, M. and Masselot, F. (2010). A Productivity Test of Statistical Machine Translation Post-Editing in a Typical Localisation Context. *The Prague Bulletin of Mathematical Linguistics*, 93:7–16.

Plutchik, R. (1980). A General Psychoevolutionary Theory of Emotion. *Emotion: Theory, Research and Experience*, 1(3):3–33.

Polajnar, T., Rogers, S., and Girolami, M. (2011). Protein interaction detection in sentences via Gaussian Processes: a preliminary evaluation. *International Journal of Data Mining and Bioinformatics*, 5(1):52–72.

Preoţiuc-Pietro, D. and Cohn, T. (2013). A temporal model of text periodicities using Gaussian Processes. In *Proceedings of EMNLP*, pages 977–988.

Quiñonero-Candela, J., Ramussen, C. E., and Williams, C. K. I. (2007). Approximation Methods for Gaussian Process Regression. *Large-scale Kernel Machines*, (2006):1–24.

Quiñonero-Candela, J., Rasmussen, C. E., Sinz, F., Bousquet, O., and Schölkopf, B. (2006). Evaluating Predictive Uncertainty Challenge. *MLCW 2005, Lecture Notes in Computer Science*, 3944:1–27.

Quirk, C. (2004). Training a sentence-level machine translation confidence measure. In *Proceedings of LREC*, pages 825–828.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*, volume 1. MIT Press Cambridge.

Rousu, J. and Shawe-Taylor, J. (2005). Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research*, 6:1323–1344.

Sahlgren, M. (2006). *The Word-Space Model*. PhD thesis, Stockholm University.

Saunders, C., Shawe-Taylor, J., and Vinokourov, A. (2003). String Kernels, Fisher Kernels and Finite State Automata. In *Proceedings of NIPS*.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47.

Settles, B. (2012). *Active learning*. Number January. Morgan & Claypool.

Severyn, A., Nicosia, M., and Moschitti, A. (2013). Learning Semantic Textual Similarity with Structural Representations. In *Proceedings of ACL*, pages 714–718.

Shah, K., Cohn, T., and Specia, L. (2013). An Investigation on the Effectiveness of Features for Translation Quality Estimation. In *Proceedings of MT Summit XIV*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2015). Taking the Human Out of the Loop : A Review of Bayesian Optimization. Technical Report 1, Universities of Harvard, Oxford, Toronto, and Google DeepMind.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge.

Snelson, E., Rasmussen, C. E., and Ghahramani, Z. (2004). Warped Gaussian Processes. In *Proceedings of NIPS*.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of AMTA*.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of EMNLP*, pages 1631–1642.

Specia, L. (2011). Exploiting Objective Annotations for Measuring Translation Post-editing Effort. In *Proceedings of EAMT*, pages 73–80.

Specia, L., Cancedda, N., Dymetman, M., Turchi, M., and Cristianini, N. (2009a). Estimating the sentence-level quality of machine translation systems. In *Proceedings of EAMT*, pages 28–35.

Specia, L. and Farzindar, A. (2010). Estimating machine translation post-editing effort with HTER. In *Proceedings of AMTA Workshop Bringing MT to the User: MT Research and the Translation Industry.*

Specia, L., Raj, D., and Turchi, M. (2010). Machine translation evaluation versus quality estimation. *Machine Translation*, 24(1):39–50.

Specia, L., Saunders, C., Turchi, M., Wang, Z., and Shawe-Taylor, J. (2009b). Improving the confidence of machine translation quality estimates. In *Proceedings of MT Summit XII*.

Specia, L., Shah, K., Souza, J. G. C. D., and Cohn, T. (2013). QuEst - A translation quality estimation framework. In *Proceedings of ACL Demo Session*, pages 79–84.

Strapparava, C. and Mihalcea, R. (2007). SemEval-2007 Task 14 : Affective Text. In *Proceedings of SemEval*, pages 70–74.

Strapparava, C. and Mihalcea, R. (2008). Learning to identify emotions in text. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 1556–1560.

Strapparava, C. and Valitutti, A. (2004). WordNet-Affect: an affective extension of WordNet. In *Proceedings of LREC*, pages 1083–1086.

Strapparava, C., Valitutti, A., and Stock, O. (2006). The Affective Weight of Lexicon. In *Proceedings of LREC*, pages 423–426.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of NIPS*, pages 3104–3112.

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Proceedings of AISTATS*, volume 5, pages 567–574.

Toutanova, K., Klein, D., Manning, C. D., and Yoram Sin (2003). Feature-rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *Proceedings of NAACL*, pages 252–259.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, 6:1453–1484.

Turchi, M., Anastasopoulos, A., de Souza, J. G. C., and Negri, M. (2014). Adaptive Quality Estimation for Machine Translation. In *Proceedings of ACL*, number 1, pages 710–720.

Turchi, M., Negri, M., and Federico, M. (2015). MT Quality Estimation for Computer-assisted Translation : Does it Really Help ? In *Proceedings of ACL*, pages 530–535.

Turian, J., Ratinov, L., and Bengio, Y. (2010). Word Representations: A Simple and General Method for Semi-supervised Learning. In *Proceedings of ACL*, pages 384–394.

Varian, H. (1975). A Bayesian Approach to Real Estate Assessment. *Studies in Bayesian Econometrics and Statistics in Honor of Leonard J. Savage*, pages 195–208.

Vishwanathan, S. V. N. and Smola, A. J. (2003). Fast Kernels for String and Tree Matching. In *Proceedings of NIPS*, pages 569–576.

Vlachos, A. (2008). A stopping criterion for active learning. *Computer Speech & Language*, 22(3):295–312.

Vulić, I. and Moens, M.-F. (2015). Bilingual Word Embeddings from Non-Parallel Document-Aligned Data Applied to Bilingual Lexicon Induction. In *Proceedings of ACL*, pages 719–725.

Wang, S. and Manning, C. D. (2012). Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. In *Proceedings of ACL*, volume 94305.

Williams, C. K. I. and Barber, D. (1998). Bayesian Classification with Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Yang, D. and Lee, W. S. (2009). Music emotion identification from lyrics. In *Proceedings of ISM*, pages 624–629.

Zellner, A. (1986). Bayesian Estimation and Prediction Using Asymmetric Loss Functions. *Journal of the American Statistical Association*, 81(394):446–451.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic-net. *Journal of the Royal Statistical Society*, 67(2):301–320.